

المختصر المفيد في لغة JavaScript

اعداد : أحمد إبراهيم

PHP

C#

VB

jQuery

HTML

AJAX

CSS

JAVA

ASP

C++

JavaScript



فهرس الكتاب

3 المقدمة
4 البداية مع JavaScript
6 الفصل الأول (الكائن window والكائن document ومفهوم الـ DOM)
6 أولاً :- الكائن window
13 ثانياً :- الكائن document
23 ثالثاً :- مفهوم الـ DOM
35 الفصل الثاني (المتغيرات variables)
40 الفصل الثالث (جمل التحكم والتكرار)
45 الفصل الرابع (المصفوفات Array)
51 الفصل الخامس (الدوال functions)
54 الفصل السادس (البرمجة كائنية التوجه oop)
54 أولاً :- الكائن النصي string object
60 ثانياً :- الكائن العددي number object
60 ثالثاً :- كائن التاريخ date object
63 رابعاً :- الكائن XMLHttpRequest
69 الفصل السابع (كائنات وأوامر منوعة)
78 الفصل الثامن (تجنب الأخطاء والأنماط)
78 أولاً :- تجنب وقوع الأخطاء
80 ثانياً :- الأنماط
88 الفصل التاسع (النماذج والإطارات)
88 أولاً :- النماذج
100 ثانياً :- الإطارات
102 الفصل العاشر (الأحداث events)
108 ملاحظات عامة

المقدمة

في بداية مسيرتي لتعلم البرمجة أحببت أن أتعلم برمجة وتصميم الموقع وبالفعل بدأت بتعلم البرمجة وأعطيتها كل وقتي وبدأت أحب البرمجة أكثر وأكثر مع العلم أنني لست بصاحب اختصاص بل شخص هاوي . في البداية بدأت بتعلم لغة HTML (وهي الأساس في تعلم تصميم المواقع ولا يمكن تصميم موقع بدونها) ثم بدأت بتعلم CSS وبعدها تعلمت لغة PHP وبعدها تعلمت لغة JavaScript , في البداية من تعلمي كنت أواجه مشكله وهي أنه لا توجد كتب أو دروس تعليمية أو أي محتوى عربي يغطي لغة من اللغات بشكل كامل بل كل ما هو موجود هو عبارة عن دروس في بعض المنتديات او على اليوتيوب وبعض الكتب الصغيرة بمعنى أن المعلومات متناثرة لذلك كنت أقرأ كتب وأحضر دروس كثيرة فقط لإضافة معلومة واحدة جديدة الى معلوماتي السابقة لأن أكثر المحتوى العربي يركز على الأساسيات ولا توجد دروس متقدمة , لذلك ولكي يسهل علي الرجوع للمعلومات التي أتعلمها عند نسيانها بدأت أدون كل ما أتعلمه وأقرأه في كتاب ليكون هذا الكتاب مرجع بسيط خاص بي , وبعد فترة من الزمن تكون لدي عدة كتب (في الـ HTML وفي الـ CSS وفي JavaScript وفي الـ PHP بالإضافة الى قواعد البيانات MySQL والتي دمجتها مع كتاب الـ PHP) وفيها قدر لا بأس به من المعلومات , في البداية لم أكن أفكر بنشر هذه الكتب بل كنت أكتبها فقط لنفسي لكن في الآخر فكرت بأنها قد تفيد البعض وتسهل عليهم الطريق , والكتب التي كتبتها هي :-

- 1- كتاب المختصر المفيد في لغة HTML - ويمكنك تحميله من [هنا](#) .
- 2- كتاب المختصر المفيد في CSS - ويمكنك تحميله من [هنا](#) .
- 3- كتاب المختصر المفيد في لغة PHP - ويمكنك تحميله من [هنا](#) .
- 4- كتاب المختصر المفيد في لغة JavaScript - وهو بين يديك الآن .

ولاحظ بأن كل هذه الكتب التي ذكرتها هو إنني قد نشرتها كما كتبتها لنفسي بدون أي تعديل او إضافة (لضيق وقتي) لذلك قد يجد البعض صعوبه في فهم بعض من نواحي هذا الكتاب (خصوصاً الذي ليس لديه أي معلومات مسبقة عن هذه اللغة) لأنني أختصرت كثيراً في توضيح الأوامر ولم أضع أمثلة مطولة بل أمثلة بسيطة فقط لتصل الفكرة لذا إذا واجهت صعوبه في فهم أي شيء لا تتردد في التواصل معي على الإيميل الموجود في أسفل الصفحة وطرح السؤال أو الإستفسار كما إنني أرحب بكل اقتراحاتكم وإذا كانت لديك أي معلومة حول هذه اللغة ولم تجدها في هذا الكتاب فإعلم أنني لا أعرفها لذا سأكون شاكراً لك إذا بعثتها لي على الإيميل الموجود في الأسفل , لاني لازلت أتعلم أشياء جديدة كل يوم الى حد هذه اللحظة والبرمجة لا تقف عند حد معين من التعلم .

هذه هي الطبعة الأولى من الكتاب وقد أنشر الكتاب كطبعة ثانية إذا أضفت اليه معلومات جديدة وقد أجري عليه بعض التعديلات وأوضح بعض الأمور الغامضة .

أحمد إبراهيم

الطبعة الأولى

ahmedcial7@gmail.com

20 - 04 - 2014

البدائية مع JavaScript

ملاحظة/ يفترض بالقارئ أن يعرف لغة HTML قبل قراءة هذا الكتاب .

طرق كتابة نص JavaScript

1. يكتب نص JavaScript داخل ملف HTML ويمكن أن يكتب الكود في القسم <body> وبهذا سينفذ مباشرة أو يمكن كتابة الكود في الوسم <head> والإشارة إليه في <body> وبذلك سينفذ الكود بعد حدوث أمر ما كضغط المستخدم على زر مثلا ويكتب كود JavaScript بين هذين الوسمين

```
<script type="text/JavaScript">
```

```
.  
.
```

```
</script>
```

* او يمكن أن يكتب كود JavaScript بين هذين الوسمين

```
<script LANGUAGE=" JavaScript">
```

```
.  
.
```

```
</script>
```

🚩 **ملاحظة//** لغة JavaScript لا تدعمها جميع المتصفحات (خصوصاً القديمة لكن أكثر المتصفحات الحديثة تدعمها , وأحيانا يكون المستخدم هو من عطل إمكانية العرض بلغة JavaScript) وإذا تم عرض الصفحة في متصفح لا يدعم لغة JavaScript لأي سبب كان فإنه سوف يعرض الكود البرمجي كما هو بدون إجراء تأثير الكود أي سيظهر الكود في الصفحة وهذه مشكله كبيرة ولأجل تلافي هذه المشكله نكتب الكود بين علامتي تعليق html

مثال/

```
<script LANGUAGE="JavaScript">
```

```
<!--
```

```
كود جافا سكريبت
```

```
//-->
```

```
</script>
```

2. كتابة كود JavaScript في ملف خارجي ونحفظه بالامتداد (name.js) مع ملاحظة عدم كتابة وسم البداية والنهاية في الكود الخارجي للجافا سكريبت بل نكتب الأكواد (الدوال function) مباشرة ، ومن ثم نستدعيه في داخل ملف HTML عن طريق هذا الكود

```
<script src="name.js"> </script>
```

الذي يكتب في الوسم <head> وبعد ذلك نستطيع أن نستدعي أي دالة نحتاجها وكأنما كود JavaScript مكتوب كله في الوسم <head> وبذلك نستطيع أن نكتب صفحة واحدة للجافا سكريبت ونشتملها في كل صفحات الموقع وفي كل صفحة نستدعي فقط الدوال التي نحتاجها بشكل اعتيادي (بذكر أسمائها) .

الفصل الأول

(الكائن WINDOW والكائن DOCUMENT ومفهوم الـ DOM)

أولاً :- الكائن WINDOW

يشير هذا الكائن إلى النافذة المفتوحة , يعتبر هذا الكائن هو الكائن الأب لجميع الكائنات الموجودة في لغة JavaScript وقبل كتابة اسم أي كائن يمكن كتابة اسم الكائن window لكن بما أنه الكائن الأب لجميع الكائنات الأخرى فيمكن عدم كتابته لأنه يمثل القيمة الافتراضية .

أ- أوامر الكائن WINDOW

alert

تستخدم هذه الدالة لإظهار رسالة تحذير للمستخدم .

مثال /

```
alert ( " This is a message " ) ;
```

او يمكن أن نكتبها مسبقة باسم الكائن الذي يحتويها وهو (window)

مثال/

```
window.alert ( " Welcome " ) ;
```

* ولا يوجد فرق بين الطريقتين .

prompt

وهي تستخدم لتظهر للمستخدم نافذة وفيها مربع إدخال نصي يمكنه من خلاله إدخال معلومات ما

مثال /

```

window.prompt ( " اسمك " , " من فضلك ادخل اسمك " ) ;

```

القيمة الابتدائية النص الذي سيظهر فوق مربع الإدخال

* يمكن أن لا نكتب اسم الكائن (window) قبل اسم الدالة prompt ولا يوجد فرق بين الطريقتين .

مثال /

```

x = prompt ( " " , " أدخل اسمك " ) ;

```

حيث في هذا المثال خزنا القيمة المدخلة في المتغير x , إذا ضغط المستخدم على إلغاء الأمر فسيعود المتغير لنا بالقيمة null .

confirm ♦

وهي تستخدم لإظهار نافذة للمستخدم لسؤاله عن قبول او رفض شيء ما , وإنها ستعيد القيمة true إذا ضغط على موافق أما إذا ضغط على إلغاء فستعيد القيمة false .

مثال /

```

confirm ("Are you sure you want to save changes ?" ) ;

```

* يمكن كتابتها مع ذكر الكائن الذي يحتويها وهو (window) , ولا يوجد فرق بين الطريقتين

مثال /

```

window.confirm ("Are you sure you want to save changes ?" ) ;

```

مثال /

```

<html> <head>
<script LANGUAGE="JavaScript">
function con() {
var c=window.confirm("Are you sure you want to save changes ?") ;
if(c) {
alert("لقد اخترت موافق");
} else {
alert("لقد اخترت إلغاء");
} }
</script> </head> <body>

```

```
<form>
<input type="button" onclick="con()" value="confirm" />
</form>
</body> </html>
```

◆ window.print ()

يستخدم هذا الأمر لإظهار النافذة الخاصة بإعطاء أمر الطباعة للطباعة لطباعة كل محتويات الصفحة الحالية المفتوحة من نصوص وصور وأزرار وغيرها , وهو مشابه لأمر الطباعة الموجود في المتصفح في قائمة الأدوات .

مثال/ لإنشاء زر يعطي أمر الطباعة

```
<input type="button" onclick="window.print ( ) ;" />
```

◆ open

وهو أحد أوامر الكائن window , ومن خلاله نستطيع فتح صفحة او نافذة جديدة , والصيغة العامة له هي :-

```
;( "الخصائص التي ستطبق على النافذة" , "اسم النافذة" , "عنوان النافذة التي ستفتح" ) window.open
```

لاحظ أنه يمكن إعطاء أي اسم للنافذة حيث يمكن استخدام هذا الاسم فيما بعد إذا احتجناه .

والخصائص التي تستخدم معه هي :-

- | | |
|---------------|---|
| 1 . top | بعد النافذة عن الجهة العلوية . |
| 2 . left | بعد النافذة عن جهة اليسار . |
| 3 . width | تحدد عرض النافذة . |
| 4 . height | تحدد ارتفاع النافذة . |
| 5 . resizable | تسمح بالتحكم في تصغير النافذة او تكبيرها القيمة yes او no . |
| 6 . toolbar | تتحكم بشريط التمرير السفلي القيمة yes او no . |
| 7 . status | للتحكم بشريط الحالة والذي يكون بالأسفل وتأخذ القيمة yes او no |
| 8 . scrollbar | للتحكم بشريط التمرير والذي يكون بجانب النافذة . |
| 9 . menubar | للتحكم بخصائص القائمة للنافذة وتأخذ القيمة yes او no . |

مثال/

```

<html>
<head>
<script LANGUAGE="JavaScript"> <!--
function op () {
var mofak ;
mofak = window.open ("http://google.com" , "name" , "top=100 , left=150 ,
width=200 , height=200 , toolbar=yes , status=no , menubar=no , scrollbars ,
resizable=yes") ;
mofak.document.write ("<body bgcolor='#ccc' text='#fff '>");
mofak.document.write ("<font size='10'> <b> google .com </b> </font>"); }
//--> </script>
</head>
<body>
<form>
<input type="button" onclick="op () " />
</form>
</body>
</html>

```

* لفتح رابط من داخل نفس الرابط في صفحة جديدة نستخدم الغرض `this.href` , ولكي لا يتم فتح الرابط في نفس النافذة الموجودين فيها نستخدم الأمر `return false` , لاحظ هذا المثال /

```

<html>
<head>
</head>
<body>
<a href=" http://google.com "
onclick="window.open ( this.href , 'popupwindow' , 'width=400 , height=300 ,
scrollbars , resizable ' ) ; return false ;" >
Click here for the link to open in a popup window.
</a>
</body>
</html>

```

close ◆

هو أحد أوامر الكائن window , وهو يستخدم لغلق النافذة الحالية المفتوحة .

مثال/

```
<input type="button" onclick="window.close ( ) ;" />
```

● moveTo

هو أحد أوامر الكائن window , وهو يستخدم لتحريك النافذة والصيغة العامة له هي :-

```
window.moveTo ( بعد النافذة عن الجهة العلوية , بعد النافذة عن جهة اليسار ) ;
```

مثال/ ينشأ زر يحرك النافذة

```
<input type="button" onclick="window.moveTo(50 , 90) ; " />
```

● resizeTo

هو أحد أوامر الكائن window , وهو يستخدم لتغيير حجم النافذة والصيغة العامة له هي :-

```
window.resizeTo ( الارتفاع , العرض ) ;
```

مثال/ ينشأ زر يغير حجم النافذة

```
<input type="button" onclick="window.resizeTo(500 , 400) ; " />
```

ب- خصائص الكائن WINDOW

● status

تستخدم هذه الخاصية لإظهار رسالة إلى المستخدم وتظهر الرسالة في أسفل المتصفح في شريط الحالة (status bar)

مثال/

```
window.status = " the message " ;
```

ت- الكائنات الفرعية في الكائن WINDOW

◆ location

وهو كائن فرعي في داخل الكائن window , وهو مسئول عن تخزين معلومات عن الـ url للصفحة الحالية وله عدة خواص تستخدم معه , لو افترضنا أن رابط الصفحة الحالية (url) هو <http://www.php.com/subject.php?sub=25>

◆ خواص الكائن location

1. location.protocol تجلب اسم البروتوكول وهو هنا http , يمكن أن نذكر اسم الكائن document قبلها أو لا نذكره .

مثال/

alert (location.protocol) ;

2. location.hostname تجلب اسم الموقع وهو هنا www.php.com , يمكن أن نذكر اسم الكائن document قبلها أو لا نذكره .

مثال/

alert (location.hostname) ;

3. location.port تجلب رقم منفذ المستخدم , يمكن أن نذكر اسم الكائن document قبلها أو لا نذكره .

مثال/

alert (location.port) ;

4. location.pathname تجلب اسم الصفحة وهو هنا subject.php , يمكن أن نذكر اسم الكائن document قبلها أو لا نذكره .

مثال/

alert (location.pathname) ;

5. location.search الاستعلام query # إذا وجد , يمكن أن نذكر اسم الكائن document قبلها أو لا نذكره .

مثال/

alert (location.search) ;

6. location.reload () تستخدم لعمل زر تحديث للصفحة (refresh) , يمكن أن نذكر اسم الكائن document قبلها او لا نذكره .

مثال/

<input type="button" value="refresh" onClick="location.reload () ;" />

7. location.replace () تستخدم للذهاب إلى صفحة جديدة , ويكتب اسم الصفحة الجديدة بين قوسي الخاصية وبين علامات تنصيص , يمكن أن نذكر اسم الكائن document قبلها او لا نذكره .

مثال/

<input type="button" value="Go " onClick="location.replace ('google.com');"/>

ملاحظة // يمكن أن يكتب اسم الكائن document قبل اسم هذه الخاصية ويسند إليه رابط صفحة جديدة ليتم الانتقال إليها

مثال/

document.location=" google.com " ;

8. location.href تستخدم هذه الخاصية لنقلنا إلى صفحة ويب جديدة وهي مشابهة لعمل الخاصية السابقة .

مثال /

<input type="button" value="Go " onClick="location.href='google.com' ;"/>

ثانياً :- الكائن DOCUMENT

يشير هذا الكائن إلى الصفحة الحالية وهو كائن فرعي للكائن الأب window

أ- أوامر الكائن DOCUMENT

◆ الأمر write

هذا الأمر هو أحد أوامر الكائن document , وهو يستخدم لطباعة نص وإظهاره على الشاشة

مثال /

```
<html>
<head></head>
<body>
<script type="text/JavaScript">
<!--
document.write("Hello");
//-->
</script>
</body> </html>
```

* يمكن طباعة قيمة متغير مسبق بجملة نصية هكذا :-

```
document.write ( " Hello " + x );
```

* ويمكن إضافة وسوم HTML في داخل أمر الطباعة لإضافة تأثير هذه الوسوم عليه , وكذلك يمكن إضافة وسوم الصور او الصوت او أي من وسوم لغة html ليتم عرضه .

مثال / لإظهار رسالة تأكيد لعملية الحذف

```
<html>
<head>
<script LANGUAGE="JavaScript">
<!--
document.write ("<font color='red' > هل أنت متأكد من إتمام عملية الحذف </font>");
document.write ("<p>");
```

```
document.write ("<center>" + "<input type='button' value='موافق' >");  
document.write ("<input type='button' value='غير موافق' ></center>");  
document.write ("</p>");  
//-->  
</script>  
</head>  
</html>
```

◆ الأمر writeln

هذا الأمر هو أحد أوامر الكائن document , وهو يستخدم للطباعة ويختلف عن الأمر السابق بأنه يترك سطر فارغ بعد النص المطبوع .

مثال /

```
document.writeln ( " Hello " );
```

* ويمكن إضافة وسوم HTML في داخل أمر الطباعة لإضافة تأثير هذه الوسوم عليه .

مثال /

```
document.writeln("Hi <br /> <font size='15'> Ali </font> ");
```

◆ lastModified

وهو أحد أوامر الكائن document ويستخدم لعرض آخر مرة تم تعديل الصفحة فيها .

مثال /

```
document.write ( document.lastModified );
```

◆ URL

وهو أحد أوامر الكائن document ويستخدم لجلب عنوان (الرابط) للصفحة .

مثال /

```
document.write ( document.URL );
```

forms ◆

وهو أحد أوامر الكائن document وهذا الأمر مسئول عن النماذج الموجودة في الصفحة ومن خلاله نستطيع أن نصل إلى النماذج والتحكم بها والعمل عليها , وسيتم شرحه بشكل مفصل مع النماذج .

title ◆

وهو أحد أوامر الكائن document , وهو يستخدم لجلب عنوان الصفحة الحالية الموجودة في الوسم <title> .
مثال /

```
alert(document.title);
```

links ◆

وهو أحد أوامر الكائن document ويستخدم لإجراء تغيير أو تأثير أو جلب معلومات عن الروابط الموجودة في الصفحة .
مثال/ لجلب عدد الارتباطات الموجودة في الصفحة

```
document.links.length ;
```

مثال/ تغيير عنوان رابط إلى عنوان آخر عند الضغط على زر معين

```
<body>
<a href="www.yahoo.com"> Yahoo </a>
<form>
<input type="button" onclick="document.links[0].href='google.com' " />
</form>
</body>
```

في داخل الزر كتبنا links وأعطيناه الرقم 0 وذلك لأن الرابط هو أول رابط في الصفحة ولو كان تسلسله ثاني رابط سنعطيه الرقم 1 وإذا كان الثالث نعطينه الرقم 2 وهكذا , او يمكن بدلاً من ذلك أن نعطي اسم للرابط من خلال الخاصية name ونذكر الاسم بدلاً من كلمة links[0] .

execCommand ◆

تستخدم هذه الدالة لإعطاء أمر ما مثل حفظ الصفحة وستظهر نافذة الحفظ وهي نفس العملية لو أننا ذهبنا إلى خيارات المتصفح وأعطيناها حفظ , او يمكن أن تعطي هذه الدالة أمر الفتح وستظهر لنا نافذة الفتح وهي نفس العملية لو أننا ذهبنا إلى خيارات المتصفح وأعطيناها فتح او ضغطنا على (ctrl + O) , ويمكن أن تعطي أمر النسخ او القص او اللصق وذلك بالتعاون مع الدالة clipboard .

مثال/ لفتح صفحة

```
<body>
<script LANGUAGE="JavaScript"> <!--
    function test ( ) {
        document.execCommand ("open") ; }
// --> </script>
<form>
<input type="button" value="open" onclick="test()" />
</form>
</body>
```

مثال/ لحفظ الصفحة

```
<body>
<script LANGUAGE="JavaScript"> <!--
    function test ( ) {
        document.execCommand ("SaveAs") ; }
// --> </script>
<form>
<input type="button" value="SaveAs" onclick="test()" />
</form>
</body>
```

الصيغة العامة للدالة execCommand هي

Object.execCommand(A , B , C) ;

ولاحظ ان هذه الدالة تعيد القيمة true اذا تمت العملية بنجاح وتعيد القيمة false اذا فشلت العملية .

A :- ويعبر عن طبيعة الأمر المعطى، وهناك لائحة من الأوامر التي يمكن تمريرها عبر هذا المتحول سنتحدث عنها بعد قليل. مثلا لو مررنا bold فالنص المختار سيصبح عريضا .

B :- قيمة منطقية تعبر عن فيما إذا توجب على التعليمة إظهار واجهة إدراج بارامترات إضافية للمستخدم، فمثلا في حال إدراج صورة ستظهر نافذة تطلب إدخال مسار الصورة فيما لو إسندت القيمة 1 الى **B** , وقد يتطلب المبرمج ان تكون واجهة المستخدم ذات خيارات أوسع ، أو أن لا تكون هناك خيارات (مثلا إدراج الايقونات الباسمة smilies لا تتطلب ادراج مسار لان مسارها معرف سابقا) فيمنع ظهور واجهة الاستفسار هذه .

C :- وهي القيمة المسندة للتعليمة في بعض الحالات، فمثلا لو اسندنا القيمة FontName الى **A** يمكننا اسناد اسم الخط "Traditional Arabic" **C** .

مثال/

document.execCommand('insertimage' , false, imgSrc) ;

حيث ان imgSrc هو مسار الصورة .

أهم البارامترات التي يمكن إسنادها للمتحول **A** هي :-

1- ForeColor : لون الخط.حيث يمكن عبر هذه التعليمة معرفة لون خط النص المحدد أو تغييره .

2- BackColor : لون تظليل الخط.حيث يمكن عبر هذه التعليمة معرفة لون تظليل خط النص المحدد أو تغييره .

3- Bold : تحويل النص المحدد الى عريض .

4- Italic : تحويل النص المحدد الى عريض .

5- Underline : اضافة خط اسفل النص المحدد .

6- StrikeThrough : تضيف خط في منتصف النص المحدد (شطب النص) .

7- Copy : نسخ النص المحدد الى الذاكرة .

8- Cut

9- Paste

10- InsertHorizontalRule : ادراج خط فاصل افقي .

11- InsertImage : ادراج صورة , وباختيار قيمة B تساوي الواحد تظهر لنا واجهة يمكن عبرها تحديد المسار والنص البديل alt والحدود borders الخ .

12- InsertMarquee : تحويل النص المختار الى شريط متحرك Marquee .

13- InsertSelectDropDown : يضيف قائمة منسدلة الى المنطقة المختارة .

14- امر الطباعة .

15- Refresh

16- CreateLink : إدراج رابط إلى النص المختار، والنافذة التي تظهر في حال B = 1 ويمكن عبرها اختار نوع البروتوكول ftp , http او الخدمة mailto بالاضافة الى ادراج الرابط .

- 17 InsertUnorderedList : ادراج قائمة منقطة .
- 18 InsertorderedList : ادراج قائمة مرقمة .
- 19 formatblock : لتحديد حجم الخط h1 h6

مثال/

```
document.execCommand( 'formatblock' , false , h1 ) ;
```

- 20 fontname : تحدد نوع الخط .

```
document.execCommand( 'fontname' , false , 'Tahoma' ) ;
```

- 21 fontsize : تحدد حجم الخط وتأخذ رقم من 1 الى 7 .

مثال/

```
document.execCommand( 'fontsize' , false , 5 ) ;
```

- 22 undo : للتراجع خطوة للخلف
- 23 redo : للتقدم خطوة للأمام .

مثال/

```
document.execCommand( "redo" , false , null ) ;
```

- 24 justifyleft : تستخدم لمحاذاة النص الى اليسار .
- 25 justifycenter : تستخدم لمحاذاة النص الى المنتصف .
- 26 JustifyFull : تستخدم لتجعل نهايات الاسطر متساوية .
- 27 justifyright : تستخدم لمحاذاة النص الى اليمين .

مثال /

```
document.execCommand( "justifyright" , false , null ) ;
```

- 28 outdent : تحرك النص مسافة (كم فاصلة tab) باتجاه اليسار .
- 29 indent : تحرك النص مسافة (كم فاصلة tab) باتجاه اليمين .
- 30 insertimage : تدرج صورة ويجب ان نضيف مسار الصورة imgSrc

مثال/

```
document.execCommand( 'insertimage' , false , imgSrc ) ;
```

- 31 removeformat : تحذف كل التنسيقات التي اجريناها من النص المحدد .
- 32 delete : تحذف النص المحدد .
- 33 DirRTL : تجعل اتجاه النص من اليمين الى اليسار .
- 34 DirLTR : تجعل اتجاه النص من اليسار الى اليمين .
- 35 Subscript : يصغر النص وينزله الى الاسفل قليلاً .
- 36 Superscript : يصغر النص ويرفعه الى الاعلا قليلاً .

مثال /

```
document.execCommand( "Subscript" , false , null ) ;
```

ملاحظة// هذه الدالة تكتب بطريقتين حسب نوع المتصفح المستخدم لاحظ هذين المثالين حيث كل منهم يعمل على بعض المتصفحات ولا يعمل على غيرها

-1

```
<html>
<head>
<script language="JavaScript">
var editor ;
function Init() {
    editor = document.getElementById( 'my_id' ) ;
    editor.contentDocument.designMode = 'on' ;
}
function doBold() {
    editor.contentDocument.execCommand('bold', false, null) ;
}
</script>
</head>
<body onLoad="Init()">
<iframe id=" my_id " style="width: 100px; height:100px"> </iframe>
<input type="button" onClick="doBold()" value="Make Me Bold!">
</body>
</html>
```

-2

```
<html>
<head>
<script language="JavaScript">
function Init() {
    my_id.document.designMode = 'on' ;
}
}
```

```
function doBold() {
    my_id.document.execCommand('bold', false, null);
}
</script>
</head>
<body onLoad="Init()">
<iframe id=" my_id " style="width: 100px; height:100px"> </iframe>
<input type="button" onClick="doBold()" value="Make Me Bold!">
</body>
</html>
```

clipboard

هذه الدالة تستخدم كحافضة تحفظ في داخلها النصوص عند نسخها او قطعها ويمكن بعد ذلك لصقها وذلك بالتعاون مع الدالة السابقة execCommand وكما في الأمثلة التالية :-

مثال 1 / نسخ النص

```
<body>
<form name="ff">
<textarea name="tt"> </textarea>
<input type="button" value="copy" onclick="test()" />
</form>
<script LANGUAGE="JavaScript"> <!--
    function test () {
        document.ff.tt.focus ();
        document.ff.tt.select ();
        x = document.selection.createRange ();
        x.execCommand("Copy"); }
// --> </script>
</body>
```

مثال 2 / قطع النص

```
<body>
<form name="ff">
<textarea name="tt"> </textarea>
<input type="button" value="Cut" onclick="test()" />
</form>
<script LANGUAGE="JavaScript"> <!--
    function test () {
```

```

document.ff.tt.focus ( ) ;
document.ff.tt.select ( ) ;
x = document.selection.createRange ( ) ;
x.execCommand("cut") ; }
// --> </script>
</body>

```

مثال 3 / لصق النص

```

<body>
<form name="ff">
<textarea name="tt"> </textarea>
<input type="button" value="Paste" onclick="test()" />
</form>
<script LANGUAGE="JavaScript"> <!--
function test ( ) {
document.ff.tt.focus ( ) ;
document.ff.tt.select ( ) ;
x = document.selection.createRange ( ) ;
x.execCommand("Paste") ; }
// --> </script>
</body>

```

ب- خصائص الكائن DOCUMENT

bgcolor ◆

هذه الخاصية هي إحدى خصائص الكائن document , وهي تستخدم لتحديد لون خلفية الصفحة .

مثال/

```
document.bgcolor = "#e9e9e9" ;
```

fgcolor ◆

هذه الخاصية هي إحدى خصائص الكائن document , وهي تستخدم لتحديد لون النص في الصفحة .

```
document.fgcolor = "#ccc" ;
```

linkColor ◆

هذه الخاصية هي إحدى خصائص الكائن document , وهي مسؤولة عن تغيير لون الروابط الموجودة في الصفحة .

مثال/

```
document.linkColor="red" ;
```

alinkColor ◆

هذه الخاصية هي إحدى خصائص الكائن document , وهي مسؤولة عن تغيير لون الروابط التي تم النقر عليها (النشطة) الموجودة في الصفحة .

مثال/

```
document.alinkColor="#ccc" ;
```

onmousemove ◆

وهو أحد خصائص الكائن document , وهو يستخدم لتطبيق تأثير معين عند المرور على صفحة , والصيغة العامة له هي :-

```
document.onmousemove = ( اسم الدالة التي تحمل التأثير ( الوظيفة ) =
```

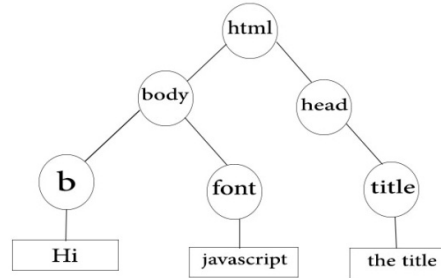
ثالثاً :- مفهوم الـ DOM

DOM وهي اختصار لـ Document Object Model , ومن خلاله يمكن التعامل بشكل كائني , لاحظ انه عندما يتم تحميل الموقع على متصفح الزائر فإن اول شيء يقوم المتصفح بقراءة الكود البرمجي المكتوب بلغة html ومن ثم يرسم هذا الكود على شكل شجرة tree وبعد ذلك يقوم المتصفح بمعالجة البيانات في الشجرة ويحولها الى الشكل الذي سيظهر به الموقع على الشاشة , ان وظيفة جافا سكربت هو التعامل مع هذه الشجرة من حيث الحذف او الاضافة او التعديل وذلك اثناء تشغيل البرنامج اي بدون الحاجة لعمل اعادة تحميل للكود البرمجي وجلب المحتويات من الخادم الى متصفح الزائر مرة أخرى .

الشجرة tree تتكون من مجموعة من الـ node وكل node يمثل عنصر من عناصر الـ htm , لاحظ هذا المثال :-

```
<html>
<head>
<title> The title </title>
</head>
<body>
<b> Hi </b>
<font size="3"> JavaScript </font>
</body>
</html>
```

سترسم الشجرة للمثال السابق بهذا الشكل



يسمى الـ node (html) أب (parent) للـ node (body) وايضاً أب (parent) للـ node (head) وكلاً من الـ body و الـ head هما ابناء (children) للـ node (html) وايضاً في نفس الوقت يعتبر الـ node (body) والـ node (head) اخوان (sibling) فيما بينهم , وكما تلاحظ فإن للـ node (body) ابناء (children) هما b و font اي انه أب لهما

وهكذا بالنسبة لكل الـ nodes في كامل الشجرة , ويسمى أول عنصر (node) في الشجرة root ويسمى اخر عنصر (node) في الشجرة أي الذي لا يتفرع منه عنصر آخر بـ leaf . لاحظ ان كل node هو عبارة عن كائن Object وهناك عدة أنواع من الـ node منها :-

1- element وهو عبارة عن وسم html

2- Attribut وهو عبارة عن خاصية

3- text وهو عبارة عن نص

وكل نوع من هذه الانواع يحمل كائنات خاصة يمكن من خلالها انشاء هذه الـ node او التعامل معها , واهم node هو الـ element وهو يحمل الكثير من الصفات منها صفات الـ CSS ومنها الاحداث ومنها التأثيرات .

وللتعامل مع أي node يجب أولاً إيجادها (Search) ثم بعد ذلك نطبق عليها التأثير الخاص بها والذي نريد تطبيقه , وكل هذا يجري من خلال كائنات ونفصل بين كل كائن وآخر بنقطة ثم نكتب اسم الكائن الآخر وهكذا .

لاحظ ان الكائن الرئيس الذي يمثل النافذة هو window وهو يكتب قبل كل الكائنات لكن بما انه الكائن الرئيسي قبل كل الكائنات فيمكن عدم كتابته لان المتصفح سيفهمه بشكل افتراضي , وهناك كائن مهم جداً وهو document وهو يكتب قبل أي كائن نريد نكتبه وكان الامر متعلق بالصفحة المفتوحة .

كائنات البحث والتحديد

body ◆

يستخدم هذا الكائن لتحديد الوسم body وعند تحديده يمكن اضافة وسوم اليه او حذف وسوم منه وهي تقريباً نفس العملية عند تحديد الوسوم عن طريق الكائن getElementById

```
var b = document.body ;
```

getElementById ◆

يعتبر هذا الكائن مهم جداً لأنه من خلاله يمكن أن نتعامل مع عنصر معين من خلال الـ id الخاص به .

مثال/

```
document.getElementById('firsr')
```

```
<div id=" first "> </div>
```

getElementsByTagName ◆

يستخدم هذا الكائن لجلب الوسوم بحسب اسمائها ويخزن الناتج على شكل مصفوفة وبذلك يمكن ان نستخدم دوال المصفوفات للتعامل معها .

مثال/

```
<p>Hello</p>
```

```
<p>Welcome</p>
```

```
.....
```

```
var tag = getElementsByTagName("p");
```

```
var text = tag[0].innerText ;
```

```
alert( text );
```

سيكون الناتج Hello

getElementsByClassName ◆

يستخدم هذا الكائن لجلب الوسوم بحسب اسماء الكلاسات ويخزن الناتج على شكل مصفوفة وبذلك يمكن ان نستخدم دوال المصفوفات للتعامل معها .

مثال/

```
<p class="js">Hello</p>
```

```
<h1 class="js">Welcome</h1>
```

```
.....
```

```
var tag = getElementsByClassName("js");
```

```
alert( tag[0].innerText );
```

سيكون الناتج Hello

nextSibling ◆

يستخدم هذا الكائن للانتقال الى العنصر الذي يلي العنصر المحدد بشرط ان يكون العنصران اخوان في الشجرة tree بمعنى ان لهم نفس الاب

مثال/

```

<div>
  <p id="pp"> Welcome </p><u> JavaScript </u>
</div>
.....
var p_node = document.getElementById(" pp ");
var u_node = p_node.nexSibling ;
alert( u_node.innerHTML ) ;

```

في هذا المثال ستظهر لنا كلمة JavaScript

ملاحظة// اذا كنا نريد ان ننتقل من عنصر الى عنصر اخ له بواسطة هذا الكائن وكان بين هذين العنصرين فراغ (مسافة) فإن جافاسكربت ستعتبر ان هذه المسافة هي عبارة عن نص وهذا النص هو عنصر اخ للعنصرين أي بمعنى ان العنصر الاول يليه عنصر ثاني هو نص فراغ وبعده العنصر الثالث .

previousSibling ◆

هذا الكائن هو يعمل تماماً عكس الكائن السابق حيث انه سيحدد العنصر الذي يسبق العنصر المحدد مع ملاحظة ان العنصرين يجب ان يكون لهم نفس الاب في الشجرة tree

parentNode ◆

من خلال هذا الكائن يمكن الانتقال (تحديد) من العنصر الابن الى العنصر الاب له في الشجرة tree

مثال/

```

<div>
  <p id="pp"> Welcome </p>
</div>
.....
var p_node = document.getElementById(" pp ");
var parent = p_node.parentNode ;
alert( parent.nodeName ) ;

```

ستظهر كلمة div

childNodes ◆

يستخدم هذا الكائن للانتقال من العنصر الابن الى العنصر الابن لكن بما ان الابناء قد يكونون اكثر من واحد لذلك فهذا الكائن يعامل كمصفوفة ونضع رقم العنصر المطلوب بين قوسى المصفوفة (وكما تعلم فان المصفوفة تبدأ من الرقم صفر)

مثال /

```
<div id="d">
<p> Welcome </p>
<i> JavaScript </i>
</div>
.....
var parent = document.getElementById(" d ");
var p_node = parent.childNodes[0] ;
alert( parent.innerHTML ) ;
```

في هذا المثال سيظهر لنا كلمة Welcome

انشاء عقدة (وسم html)**createElement** ◆

يستخدم هذا الكائن لإنشاء وسم html جديد (ولاحظ ان العقدة node التي سنتنشأ في الشجرة من نوع وسم element)

مثال/

```
var x = createElement("b");
x.innerHTML("Welcome");
```

في هذا المثال انشأنا الوسم وكتبنا بداخله Welcome لكن لاحظ اننا انشأنا هذا الوسم لكن لم نحدد مكان معين ليظهر فيه ولتحديد مكان معين يجب ان نستخدم الكائن appendChild انه سيضيف هذا الوسم الجديد كأبن لوسم اخر وهذا الوسم الاخر سنحدده من خلال الخاصية id

مثال/

```
<h1 id="we"></h1>
....
var x = createElement("b");
x.innerHTML("Welcome");
var we_js = document.getElementById("we");
we_js.appendChild(x);
```

createTextNode ◆

يستخدم هذا الكائن لإنشاء عقدة (عنصر) node في الشجرة لكن هذه العقدة ستكون من النوع نص , ونضع بين قوسي هذا الكائن النص .

مثال/

```
<h1> </h1>
.....
var t = document.createTextNode("the text");
alert(t.nodeValue);
var h = document.getElementsByTagName("h1");
h[0].appendChild( t ) ;
```

appendChild ◆

يستخدم هذا الكائن لإضافة وسم ابن للوسم المحدد (تحديد الوسم الابن)

مثال

```
<h1 id="we"></h1>
....
var x = createElement("b");
x.innerHTML("Welcome");
var we_js = document.getElementById("we");
we_js.appendChild(x);
```

parentNode ◆

يستخدم هذا الكائن لتحديد الوسم الأب للوسم المحدد أي أنه عكس الكائن السابق

مثال/

```
<h1>
<p id="pp"> google.com </p>
</h1>
....
var p = document.getElementById("pp");
var node = p.parentNode ;
node.innerText("Yahoo.com");
```

في هذا المثال سيستبدل جوجل بياهو .

كائنات التعامل مع الـ nodes

nodeType ◆

يستخدم هذا الكائن ليعيد نوع العنصر المحدد ومعرفة نوع العنصر مهم جداً لمعرفة كيفية التعامل معه وأي دوال يمكن استخدامها , وهو يعيد مجموعة من الأرقام ولاحظ ان كل رقم يشير الى نوع معين :-

- 1 يشير الى ان نوع العنصر هو وسم مثل (div , p , h1)
- 2 يشير الى ان نوع العنصر هو خاصية مثل (class , border)
- 3 يشير الى ان نوع العنصر هو نص مثل (Welcome)
- 8 يشير الى ان نوع العنصر هو تعليق مثل (<!-- Comment -->)
- 9 يشير الى ان نوع العنصر هو العنصر الاساسي document

```
<p id="pp">Hello</p>
.....
var x = document.getElemenById("pp") ;
alert( x.nodeType );
```

سيعيد الرقم 1

nodeValue ◆

يستخدم هذا الكائن لجلب قيمة العنصر المحدد لكن انتبه اذا كان العنصر المحدد هو وسم element سوف تعيد القيمة null واذا كان نوع العنصر المحدد هو خاصية attribut لوسم سوف يعيد قيمة هذه الخاصية واذا كان العنصر المحدد من نوع نص text فانه سوف يعيد هذا النص .

مثال/

```
<p id="pp">Hello</p>
.....
var x = document.getElemenById("pp") ;
alert( x.nodeValue );
```

سوف يعيد القيمة null

nodeName ◆

يستخدم هذا الكائن لجلب اسم الوسم المحدد

مثال/

```
<p id="pp">Hello</p>
.....
var x = document.getElemenById("pp") ;
alert( x.nodeName );
```

getAttribute ◆

هذا الكائن يستخدم لجلب خاصية من خصائص الوسم المحدد

مثال/

```
<font id="pp" size="3"> google.com </p>
....
var p = document.getElementById("pp");
var at = p.getAttribute("size");
ater(at);
```

ستخرج لنا القيمة 3

setAttribute ◆

يستخدم هذا الكائن لاضافة خاصية جديدة او التعديل على خاصية في الوسم المحدد .

مثال/

```

<font id="pp" size="3"> google.com </p>
....
var p = document.getElementById("pp");
p.setAttribute("color","#f00");
p.setAttribute("size","5");
p.setAttribute("style","font-family:Tahoma; background-color:#000;");

```

removeAttribute ◆

يستخدم هذا الكائن لحذف خاصية من خصائص الوسم المحدد

مثال/

```

<font id="pp" size="3"> google.com </p>
....
var p = document.getElementById("pp");
p.removeAttribute(size);

```

removeChild ◆

يستخدم هذا الكائن لحذف وسم من وسوم الشجرة لكن لاحظ ان هذا الكائن يحذف العنصر الابن للعنصر الاب ولحذف عنصر ما يجب ان نحدد العنصر الاب له .

مثال/

```

<div>
<font id="pp" size="3"> google.com </p>
</div>
....
var p = document.getElementById("pp");
var d = p.parentNode("color","#f00");
d.removeChild(p);

```

clientHeight ◆

تستخدم هذه الخاصية لجلب ارتفاع العنصر المحدد .

مثال/

```
var p = document.getElementById("pp");
var h = p.clientHeight ;
alert(h);
```

value ◆

من خلال هذه الخاصية يمكن التعامل مع القيم المدخلة في حقول الإدخال في النماذج .

مثال/

```
var num = document.getElementById("num_id").value;
alert num ;
<input type="text" id="num_id">
```

innerHTML ◆

من خلال هذا الكائن يمكن أن نكتب داخل العنصر المحدد .

مثال/

```
<body>
<div id="tt"> </div>
<script LANGUAGE="JavaScript"><!--
document.getElementById("tt").innerHTML= "My Text" ;
//--> </script>
</body>
```

ويمكن كتابة المثال السابق بهذا الشكل

```
<body>
<div id="tt"> </div>
<script LANGUAGE="JavaScript"><!--
tt.innerHTML = "My Text" ;
//--> </script>
</body>
```


ويمكن أن نكتب بداخلها وسوم html او متغيرات معرفة مسبقاً لاحظ :-

```
var name_var = 55 ;
document.getElementById("tt").innerHTML= name_var + "<h1> My Text </h1>" ;
```

مثال/

```
document.getElementById("tt").innerHTML= "<img src='img.gif' border='0' />" ;
```

كما لاحظت من خلال هذا المثال يمكننا وضع أي وسم من وسوم html .
ملاحظة/ اذا لم نكتب أي قيمة بداخلها فأنها سوف تعيد لنا كود الـ html للعنصر المحدد

◆ innerText

يستخدم هذا الكائن لعرض النصوص فقط وهو مشابهة للكائن السابق لكنه لا يعرض وسوم html بل يعرض نصوص فقط .

ملاحظة/ اذا لم نكتب أي قيمة بداخلها فأنها سوف تعيد لنا النص الموجود داخل العنصر المحدد .

◆ style

من خلال هذا الكائن يمكن التحكم بجميع خصائص الـ CSS للعنصر المحدد حيث سنضع نقطة بعدها اسم الخاصية التي نريد اضافتها او التي نريد اجراء التعديل عليها وبعدها يساوي وبين علامات تنصيب نضع القيمة .

مثال/

```
document.getElementById("top").style.color = "#F00" ;
```

.....

```
<p id="top"> the text </p>
```

ملاحظة// في لغة جافاسكربت يتم كتابة الدوال بطريقة camel style وهذه الطريقة تنص على انه اذا كانت الدالة متكونة من اكثر من كلمة سيتم كتابة الكلمة الاولى باحرف صغيرة وأول حرف من كل كلمة اخرى يكتب كبير والكلمات تكون متلاصقة مع بعضها , ولكن لغة الـ CSS تفصل بين الكلمات بفاصلة (-) , ومن الكلام السابق يتبين لنا انه يجب ان نكتب خصائص الـ CSS التي تحتوي على فاصلة بطريقة الجافا سكربت أي ان نحذف الفاصلة ونحول الحرف بعدها الى كبير مثلا font-family سنكتبها بالجافا سكربت بهذا الشكل fontFamily واذا اردنا

ان نكتب background-color سنكتبها باجافاسكربت بهذا الشكل backgroundColor وهكذا بالنسبة لبقية الخصائص .

مثال/

```
document.getElementById("top").style. fontFamily = " Tahoma " ;
```

.....

```
<p id="top"> the text </p>
```

selectedIndex ◆

يستخدم هذا الكائن مع القائمة المنسدلة (select) وهو سيعتبر أن عناصر هذه القائمة عبارة عن مصفوفة حيث أنه سيعيد رقم العنصر المحدد ويبدأ العد من الصفر .

مثال/

```
function show_select() {  
var x = document.getElementById("id").selectedIndex;  
alert x ; }  
<select id="id" onChange="show_select()">  
<option >name</option>  
<option >password</option>  
<option >email</option>  
</select>
```

الفصل الثاني

(المتغيرات VARIABLES)

أسم المتغير يجب أن يبدأ بحرف أو شرطة سفلية _ ولا يمكن أن يبدأ برقم لكن يمكن أن يتضمن في داخله رقم , ولإنشاء متغير يجب أن نعرفه أولاً ونستخدم لذلك (var)

مثال /

```
var name="Ahmed" ;
```

* يمكن تعريف المتغير وإعطائه القيمة في وقت آخر , هكذا :-

```
var name ;  
name = "Ahmed" ;
```

* يمكن تعريف عدة متغيرات دفعة واحدة ونعطيها القيم في وقت آخر , هكذا :-

```
var x , y , z ;
```

* ويمكن أن نسند لعدة متغيرات قيمة واحد في سطر واحد , هكذا :-

```
var x , y , z = 10 ;
```

* يمكن إضافة متغير نصي إلى آخر بوضع إشارة (+) بينهما .

```
x = " Hi " ;  
y = " Ahmed " ;  
z = x + " " + y ;
```

*المعاملات الرياضية

المعامل	الوصف
+	الجمع
-	الطرح
*	الضرب
/	القسمة
%	باقي القسمة
++	الزيادة بمقدار 1
--	الإنقاص بمقدار 1

* معاملات التعيين

المعامل	المثال	المثال بصيغة أخرى
=	$X = y$	$X = y$
+ =	$X + = y$	$X = X + y$
- =	$X - = y$	$X = X - y$
* =	$X * = y$	$X = X * y$
/ =	$X / = y$	$X = X / y$
% =	$X \% = y$	$X = X \% y$

مدة حياة المتغير

عند الإعلان عن متغير في داخل دالة فإنه يمكن الوصول إليه فقط من داخل نفس الدالة ويسمى متغير محلي (Local Variable) أما عند الإعلان عن متغير خارج الدالة فإنه يمكن الوصول إليه من أي مكان أو داخل أي دالة ويسمى متغير عام (Global Variable) .

الدوال المستخدمة مع المتغيرات

◆ parseInt

تستخدم لتحويل المتغير إلى متغير من نوع integer (عدد صحيح) , حيث إن هذه الدالة تحول المتغير إذا كان رقم وموضوع بين علامات تنصيص " " (متغير نصي) إلى متغير رقمي , مثلاً لو أردنا أخذ قيمة رقمية من المستخدم فإنه حتى لو أدخل قيمة رقمية في الحقل النصي فإن المعالج في لغة JavaScript سيعتبرها وكأنها قيمة نصية (أي وكأنها موضوعة بين علامات تنصيص " ") ولا يمكننا إجراء العمليات الحسابية عليها من جمع أو ضرب أو غيرها من العمليات الحسابية وللتخلص من هذه المشكلة نستخدم هذا الدالة (parseInt) لتحويل القيمة إلى رقمية .

الصيغة العامة لها هي :-

; (القيمة النصية المراد تحويلها) parseInt

مثال /

```
var n = " 55 " ;
parseInt ( n ) ;
```

مثال /

```
var x = prompt ( " ادخل الرقم الأول " , " " ) ;
var y = prompt ( " ادخل الرقم الثاني " , " " ) ;
var z = parseInt ( x ) + parseInt ( y ) ;
alert ( x + " + " y + " = " z ) ;
```

👉 ملاحظة // هذه الدالة تحول القيمة إلى عدد صحيح , أي لو أردنا جمع قيمتين يدخلهما المستخدم وكانت القيم المدخلة عبارة عن أرقام عشرية واستخدمنا هذه الدالة لتحويل القيم المدخلة (التي تعتبرها JavaScript نصية) إلى قيم رقمية وجمعنا هذه القيم فإنه سيتم تجاهل الأرقام التي تقع بعد الفارزة وسيجمع فقط الأرقام الصحيحة التي تقع قبل الفارزة .

parseFloat ◆

تستخدم لتحويل المتغير النصي (القيمة المدخلة من المستخدم في الحقل النصي تعتبر قيمة نصية حتى لو كانت أرقام أي وكأنها موضوعة بين علامات تنصيص " ") إلى متغير من نوع float (قيمة رقمية عشرية)

/مثال/

```
var n = " 5.4 " ;
parseFloat ( n ) ;
```

/ مثال /

```
<html> <head>
<script LANGUAGE="JavaScript">
<!--
var name ;
name = window.prompt ( " أدخل اسمك " , " الاسم " ) ;
document.write ( " أهلاً بك " + name ) ;
//-->
</script></head>
<body></body></html>
```

Number ◆

تعمل هذه الدالة نفس عمل الدالتين السابقتين (parseInt , parseFloat) , أي أنها تحول القيمة المدخلة من المستخدم في الحقل النصي والتي ستعتبرها JavaScript قيمة نصية حتى لو كانت عبارة عن أرقام إلى قيمة رقمية , ستكون هذه القيمة الناتجة عدد صحيح إذا كانت القيمة المدخلة رقم صحيح او تكون القيمة الناتجة رقم عشري إذا كانت القيمة المدخلة هي رقم عشري أي وكأنها تجمع عمل الدالتين السابقتين معاً , والصيغة العامة لها هي :-

; (القيمة النصية المراد تحويلها إلى رقمية) **Number**

/مثال/

```
var n = " 5.43 " ;
Number ( n ) ;
```

* لاحظ أن الحرف (N) في اسم الدالة يجب أن يكون كبير .

isNaN ◆

هذه الدالة تقوم بفحص المتغير والتعرف عليه هل هو قيمة رقمية أم قيمة نصية , وتخرج قيمة الدالة في شكل نتيجة منطقية true او false , والصيغة العامة لهذه الدالة هي :-

isNaN (القيمة المراد فحصها) ;

الفصل الثالث

(جمل التحكم والتكرار)

* معاملات المقارنة

المعامل	الوصف
==	التساوي بالقيمة فقط
===	التساوي بالقيمة والبيانات
!=	عدم التساوي بالقيمة
!==	عدم التساوي بالقيمة والبيانات
>	أكبر من
<	أصغر من
>=	أكبر من أو يساوي
<=	أصغر من أو يساوي

* المعاملات المنطقية

المعامل	الوصف
&&	And (و)
	Or (او)
!	not (نفي)
?:	شرط

أولاً :- أدوات الشرط

◆ **الدالة الشرطية if**

وهي أداة تقوم بالتحقق من شرط معين إذا تحقق تنفذ أمر ما , والصيغة العامة لها هي :-

{ كود ينفذ إذا تحقق الشرط } (الشرط) if

◆ الدالة الشرطية if ... else

وهي نفس الأداة if ولكن هنا سوف ينفذ كود أيضاً في حالة عدم تحقق الشرط , والصيغة العامة لها هي :-

{ كود ينفذ إذا لم يتحقق الشرط } else { كود ينفذ إذا تحقق الشرط } (الشرط) if

مثال /

```
var x = " Ali " ;
var y = " Ahmed " ;
if ( x == y ) {
document.write ( " متساوي " ) ;
} else {
document.write ( " غير متساوي " ) ;
}
```

* يمكن استخدام جمل if بداخل بعضها البعض .

◆ الشرط المختصر : ?

وهو يعتبر نفس أداة الشرط if else لكن يكتب بسطر واحد , والصيغة العامة له هي :-

; كود عدم تحقق الشرط : كود تحقق الشرط ? الشرط

◆ الدالة الشرطية switch

هي تستخدم إذا كانت لدينا عدة احتمالات أي نستخدم في داخلها عدة شروط ويجب أن يتحقق واحد فقط , والصيغة العامة لها هي :-

```
switch ( المتغير ) {
case الاحتمال الأول :
الكود الذي سينفذ إذا تحقق الشرط ( الاحتمال الأول )
break ;
case الاحتمال الثاني :
الكود الذي سينفذ إذا تحقق الشرط ( الاحتمال الثاني )
break ;
default :
الكود الذي ينفذ إذا لم يتحقق أي شرط
}
```

مثال /

```

var contry ;
contry = window.prompt ( " أدخل الدولة لتعرف عاصمتها" ) ;
switch ( contry ) {
case " العراق " :
document.writeln ( " <h3> بغداد </h3>" ) ;
break ;
case " المغرب " :
document.writeln ( " <h3> الرباط </h3>" ) ;
break ;
default :
document.writeln ( " <h3> الدولة التي أدخلتها غير موجودة في قاعدة البيانات </h3>" ) ;
}

```

ثانياً :- حلقات التكرار

◆ حلقة التكرار for

مثال/

```

for ( var i = 0 ; i <= 10 ; i++ ) {
document.write ( i ) ; }

```

لاحظ في المثال أعلاه عرفنا المتغير i في داخل عبارة التكرار ويمكن تعريفه خارجها .

ملاحظة // يمكن عدم تعريف المتغير الذي سيكون هو بمثابة العداد داخل حلقة التكرار i وكتابته مباشراً بدون تعريف لا داخل ولا خارج الحلقة .

* يمكن أن نضع في حلقة التكرار الواحدة أكثر من عداد و سنفصل بين الواحد والآخر بفارزة ويمكن أن نزيد قيمة كل منهم على حدة ويمكن أن تكون قيمة الزيادة عبارة عن عملية رياضية .

مثال/

```

for ( i=0 , x=5 , j=3 ; i<=10 ; i++ , j=i*2 , x-- ) {
document.write ( i + j + x + "<br />" ) ; }

```

◆ الجملة التكرارية for in

تمكننا هذه الجملة التكرارية من إجراء عملية على المصفوفة بدون الحاجة لمعرفة عدد عناصرها والصيغة العامة لها هي :-

```
for ( الكود { الكائن in المتغير } )
```

مثال/

```
var Ax = new Array ( 2 , 6 , -6 , 3 , 22 , 0 , 101 ) ;
var sum = 0 ;
for ( var index in Ax ) {
sum += Ax[index] ;
document.write ( Ax[index] ) ;
document.write ( "<br />" ) ; }
document.write ( " مجموع عناصر المصفوفة " + sum ) ;
```

◆ حلقة التكرار while

مثال /

```
var i = 1 ;
while ( i <= 10 ) {
document.write ( i ) ;
i ++ ; }
```

◆ حلقة التكرار Do ... while

الصيغة العامة لها هي :-

```
do { نفذ الأمر
مقدار الزيادة
} while ( عد إلى داخل الحلقة مرة أخرى ونفذ مادام الشرط يتحقق )
```

مثال/

```
var i = 1 ;
do { document.write ( i ) ;
i ++ ;
}
while ( i <= 10 )
```

الدوال المستعملة مع أدوات الشرط وجمل التحكم

break -1

هذا الأمر يستخدم عندما نريد أن نخرج من حلقة التكرار عند وصوله لشيء معين .

مثال/

```
var x = new Array ("a" , "b" , "c" , "d" , "e") ;  
for ( var i = 0 ; i < x.length ; i ++ ) {  
if ( i == 3 )  
break ;  
document.writeln ( x [ i ] + "<br />" )
```

هنا سينفذ أمر الطباعة فقط على الثلاث عناصر الأولى وعند الوصول إلى الحلقة الرابعة يخرج من حلقة التكرار

continue -2

وهي تستخدم لتجاهل شيء معين وتستخدم داخل حلقات التكرار .

مثال/

```
var x = new Array ("a" , "b" , "c" , "d" , "e") ;  
for ( var i = 0 ; i < x.length ; i ++ ) {  
if ( i == 3 )  
continue ;  
document.writeln ( x [ i ] + "<br />" )
```

لاحظ في هذا المثال بأنه سيطبع جميع العناصر عدا العنصر الرابع أي صاحب الرقم ثلاثة في المصفوفة لأن تسلسل العناصر في المصفوفة يبدأ من الصفر .

الفصل الرابع

(المصفوفات ARRAYS)

يمكن تعريف المصفوفة مع ذكر عدد العناصر فقط وبعدها نذكر قيم العناصر .

```
var a = new Array ( 5 ) ;
```

```
a = [ 3 , 2 , 6 ] ;
```

ويمكن أن لا نذكر عدد العناصر فيكون ما بين القوسين فارغ .

او يمكن تعريف المصفوفة بشكل آخر وذلك بذكر قيم العناصر مباشرة

```
var a = new Array ( "ali" , 5 , "Ahmed" ) ;
```

او يمكن تعريف المصفوفة بشكل آخر هكذا :-

```
var a = [ "Ali" , 7 , "Ahmed" ] ;
```

ملاحظة / الترتيم في المصفوفة يبدأ من الصفر أي إن رتبة أول عنصر تكون صفر وليس واحد ورتبة العنصر الثاني تكون واحد وهكذا

دوال المصفوفات

length ◆

هذا الأمر يستخدم لمعرفة طول المصفوفة ويعود لنا بعدد عناصر المصفوفة , والصيغة العامة لها هي :-

```
اسم المصفوفة.length ;
```

مثال/

```
var x = new Array ( "Ali" , "Ahmed" , "basim" ) ;
```

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

```
document.writeln ( x[ i ] + "<br />" ) ; }
```

delete ◆

وهو يستخدم لحذف عنصر من عناصر المصفوفة .

مثال / لحذف العنصر الثالث من المصفوفة

```
var ocn = new Array ("a" , "b" , "c" , "d" , "e") ;
```

```
delete ocn[2] ;
```

* عند حذف عنصر من المصفوفة يبقى مكانه فارغ في المصفوفة ويبقى عدد عناصر المصفوفة كما هو ولا يتغير .

الدوال pop و push و shift و unShift ◆

الدالة push تستخدم لإضافة عنصر جديد في آخر المصفوفة .

الدالة pop تستخدم لحذف آخر عنصر من المصفوفة .

الدالة unShift تستخدم لإضافة عنصر في أول المصفوفة .

الدالة shift تستخدم لحذف أول عنصر من المصفوفة .

الصيغة العامة هي :-

; (قيمة العنصر الجديد) .push اسم المصفوفة

; () .pop اسم المصفوفة

; (قيمة العنصر الجديد) .unShift اسم المصفوفة

; () .shift اسم المصفوفة

مثال/

```
var arr = new Array ("a" , "b" , "c" , "d") ;
```

```
arr.push ( " z " ) ;
```

```
arr.unShift ( " a " ) ;
```

```
arr.pop ( ) ;
```

```
arr.shift ( ) ;
```

conCat () ◆

تستخدم هذه الدالة في وصل عناصر مصفوفتين مستقلتين معاً في مصفوفة واحدة .

مثال/

```
var arr1 = new Array ("A" , "B" , "C") ;  
var arr2 = new Array (5,3,9) ;  
var con = arr1.concat ( arr2 ) ;
```

slice () ◆

تستخدم هذه الدالة في تقسيم المصفوفة ونسخ جزء من هذه المصفوفة في مصفوفة جديدة ويوضع بين قوسي الدالة معاملين الأول يمثل رتبة العنصر الأول الذي تريد نسخه والمعامل الثاني يمثل رتبة العنصر الأخير الذي تريد نسخه وهو لا يدخل في عملية النسخ , او يمكن عدم كتابة العنصر الأخير أي نكتب رتبة العنصر الأول فقط وفي هذه الحالة سيأخذ من رتبة العنصر الأول الذي ذكرناه إلى نهاية المصفوفة .

مثال/ لنسخ الحرفين B و C

```
var x = new Array ("A" , "B" , "C" , "D" , "E") ;  
var y = x.slice ( 1 , 3 ) ;
```

splice () ◆

تستخدم هذه الدالة لأخذ جزء معين من مصفوفة وكتابته في مصفوفة جديدة حيث سوف نذكر بين قوسها رتبة العنصر الأول الذي نريد أن تبدأ منه المصفوفة الجديدة وبعده نكتب عدد العناصر التي ستنتقل إلى المصفوفة الجديدة .

مثال /

```
var arr = new Array ("a" , "b" , "c" , "d" , "e" , "f" , "g" , "h") ;  
var con = arr.splice ( 3 , 2 ) ;
```

◆ **join ()**

تستخدم هذا الدالة في وضع فاصلة بين قيم عناصر المصفوفة , والصيغة العامة لها هي :-

;(العلامة الفاصلة) .join اسم المصفوفة

مثال/

```
var x = new Array ( "Ali" , "Ahmed" , "basem" );
var y = x.join ("<br />");
document.write ( y );
```

مثال/

```
var x = new Array ( "Ali" , "Ahmed" , "basem" );
var y = x.join ( " , " );
document.write ( y );
```

◆ **الدالة toString و toLocaleString**

تقوم كلتا الدالتين بدمج قيم عناصر المصفوفة في متغير نصي بوضع فاصلة بين القيم , فعند استخدام الدالة toString يكون الفاصل المستخدم هو الفارزة " , " , وبذلك فإنها تكافئ عمل الدالة join كما يلي :-

;(" , ") .join اسم المصفوفة

أما عند استخدام الدالة toLocaleString يكون الفاصل المستخدم هو الفارزة المنقوطة " ; " , وبذلك فإنها تكافئ عمل الدالة join كما يلي :-

;(" ; ") .join اسم المصفوفة

والصيغة العامة لها هي :-

;() .toLocaleString اسم المصفوفة

;() .toString اسم المصفوفة

◆ **sort ()**

تستخدم هذه الدالة في ترتيب العناصر بداخل المصفوفة على حسب قيم هذه العناصر , فلو كانت العناصر بداخل المصفوفة تأخذ قيم من نفس النوع كقائمة الأسماء مثلاً او أعمار فيمكن ترتيب العناصر أبجدياً او تصاعدياً او تنازلياً .

مثال/

```
var arr1 = new Array ("c" , "a" , "b") ;
var arr2 = arr1.sort ( ) ;
alert ( arr2.join ( " \n " ) ) ;
```

مثال/

```
var names = new Array ("Ali" , "kalid" , "basem") ;
var i ;
names.sort ( ) ;
document.write ( " new names again in order " + "<br />" ) ;
for ( i = 0 ; i < names.length ; i++ ) {
document.write ( names[ i ] + "<br />" ) ; }
```

📌 **ملاحظة //** يتم ترتيب الحروف على حسب أرقامها في نظام (أسكي) من أصغر رقم إلى أكبر رقم وبذلك سيكون هناك فرق بين الحروف الكبيرة والصغيرة لأن الحروف الصغيرة تبدأ من الحرف a الذي يساوي 97 وتزداد بمقدار واحد أما الحروف الكبيرة فتبدأ من الحرف A والذي يساوي 65 وتزداد بمقدار واحد .

*إذا أردنا ترتيب عناصر مصفوفة تصاعدياً او تنازلياً فيجب أن نكون داله بهذا الخصوص ثم إسنادها إلى الدالة sort

مثال لترتيب عناصر مصفوفة تصاعدياً

```
function compare ( a , b ) {
return a - b ; }
var arr1 = new Array ( 12 , 5 , 200 , 8 ) ;
var arr2 = arr1.sort ( compare ) ;
alert ( arr2.join ( " , " ) ) ;
```

reverse ()

تستخدم هذه الدالة لعكس ترتيب العناصر في المصفوفة حيث يصبح آخر عنصر هو أول عنصر في المصفوفة .

مثال/

```
var arr = new Array ("A" , "B" , "C") ;
x.reverse ( ) ;
```

تداخل المصفوفات

لاحظ هذا المثال وهو سيوضح معنى المصفوفات المتداخلة

```
var arr = [ [1, 2, 6], ["a", "n", "t"], [2, 4, "d"] ];  
document.writeln ("<h2>طباعة مصفوفة ثنائية الأبعاد </h2>");  
for ( var i = 0 ; arr < arr.length ; ++ i ) {  
for ( var j = 0 ; j < arr.length ; ++ j ) {  
document.writeln ( arr [ i ] [ j ] + " " ); }  
document.writeln ( "<br />" ); }
```

ملاحظة/ لا يوجد فرق بين هذا :-

```
for ( var i = 0 ; i < اسم المصفوفة.length ; ++ i )
```

وهذا :-

```
for ( var i in اسم المصفوفة )
```

مثال /

```
var arr = new Array ( 2 ) ;  
arr[0] = new Array ("a", "b", "c") ;  
arr[1] = new Array ("d", "e", "f", "g") ;
```

مثال /

```
var x = new Array ("a", "b", "c") ;  
var y = new Array ("d", "e", "f", "g") ;  
var arr = new Array ( 2 ) ;  
arr[0] = x ;  
arr[1] = y ;
```

الفصل الخامس

(الدوال FUNCTIONS)

وهي دوال يتم كتابتها في الجزء <head> ويتم استدعاؤها في القسم <body> , والصيغة العامة لها :-

```
function اسم الدالة ( متغير1 , متغير2 , متغير3 , الخ ) {
الكود
return النتيجة المعادة ; }
```

* لاحظ بأنه يمكن أن لا نكتب أي متغير بين القوسين وكذلك يمكن أن لا نكتب (return) إذا لم تكن هناك قيمة معادة وعند استدعاء الدالة يتم ذكر اسم الدالة فقط مع قيم المتغيرات بين القوسين إذا كانت موجودة , وكذلك يمكن أن نكتب أمر التنفيذ في داخل الدالة وبذلك ستنفذ بمجرد استدعاؤها
مثال /

```
function result ( a , b ) {
c = a + b ;
return c ; }
x = result ( 2 , 3 ) ;
```

في هذا المثال وضعنا الدالة في المتغير (x) يتم تعريف الدالة في القسم <head> وذلك لضمان تحميل الدالة قبل استدعاؤها حيث يتم استدعاؤها في القسم <body>

* يمكن استدعاء دالة من داخل دالة أخرى وذلك بمجرد ذكر أسمها أي بنفس الطريقة التي يتم فيها استدعاء الدالة في مكان آخر .

* لاحظ بأنه إذا عرفنا متغير خارج الدالة يمكن استخدامه في داخلها (بعكس ما تعلمناه في الـ php)

جملة الإرجاع RETURN

الدوال التي سوف تعاد نتيجة بعد تنفيذها يجب أن تستخدم جملة return , هذه الجملة تحدد القيمة التي يجب أن تعاد للمكان الذي استعملت فيه .

* يمكن قطع السطر البرمجي داخل السلسلة النصية بشطرة مائلة للخلف \ , لاحظ هذا المثال /

```
document.write ( " Hello \
Ahmed " );
```

لاحظ هذا المثال الثاني فهو **خاطئ** :-

```
document.write \
( " Hello Ahmed " );
```

يمكن تعريف دوال كائنية بطريقة أخرى (أي وكأن الدالة هي كائن) لاحظ هذا المثال :-

```
var fun = function( var ){
this.name1 = "value1" ;
this.name2 = var ;
this.name3 = "value3" ;
}
```

هنا قمنا بتعريف الدالة وسميناها fun ثم لاحظ بأنه يجب وضع الكائن this قبل كل اسم قيمة ثم نضع القيمة , والآن يمكن استدعاء قيمة معينة من داخل هذه الدالة الكائنية بهذا الشكل

```
var ob = new fun( "value" ) ;
alert( ob.name2 ) ;
```

هنا قمنا بتعريف كائن جديد اسميناها ob واسندنا له الدالة الكائنية وبعدها قمنا بأظهار القيمة الثانية من الدالة .

ملاحظة/ يمكن وضع دالة function داخل دالة أخرى .

مثال /

```
var fun = function(){  
  this.name1 = "value1" ;  
  this.name2 = function() {  
    document.write( "Value" ) ;  
  } ;  
  this.name3 = "value3" ;  
}
```

ويمكن استدعاء الدالة قيم الدالة الداخلية بهذا الشكل

```
var ob = new fun() ;  
alert( ob.name2 ) ;
```

الفصل السادس

(البرمجة كائنية التوجه OOP)

أولاً :- الكائن النصي STRING OBJECT

لإنشاء كائن نصي نكتب هذا الكود :-

```
var name1=new String ( القيمة سواء كانت نص او أرقام )
```

خصائص الكائن النصي

👉 **ملاحظة //** ويمكن استخدام خواص الكائن النصي مع المتغير النصي حيث أن JavaScript ستحول المتغير النصي بشكل تلقائي إلى كائن نصي وتجري عمل الخاصية المطبقة عليه .

◆ length

للكائن النصي خاصية الطول كما هو الحال في المصفوفة وقيمة خاصية الطول في الكائن النصي عبارة عن عدد الأحرف التي يتكون منها النص (أي تعود برقم) , والكود التالي يقوم بوضع عدد الحروف التي يتكون منها الكائن (name1) كقيمة للمتغير (Lname)

```
var Lname = name1.length ;
```

◆ charCodAt و charAt

تستخدم هذه الخواص لمعرفة معلومات عن حرف واحد من الأحرف وتكمن فائدة هذه الخواص في فحص ما يدخله المستخدم في النماذج مثل البريد الإلكتروني , ويتم وضع معامل واحد فقط بين قوسيهما , حيث أن وظيفة charAt تعود برتبة الحرف داخل النص .

مثال/

```
var x = prompt ( " inter text " , "Hello !" ) ;
ver Last = x.charAt ( x.length - 1 ) ;
document.write ( " The last character is " + Last ) ;
```

لاحظ هنا أنقصنا مقدار واحد لأن آخر حرف هو علامة التعجب (!) ونحن نريد معلومات عن الحرف الأخير .

وإن وظيفة charCodAt هي نفس وظيفة charAt ولكن بدلا من استخراج الحرف نفسه تقوم باستخراج الرقم او الكود بالنظام السداسي العشري (أسكي) .

مثال/

```
var x = prompt ( " inter text " , "Hello !" ) ;
ver First = x.charCodAt ( 0 ) ;
document.write ( " The First character is " + First ) ;
```

حيث إن هذا المثال سيقوم بطباعة أول حرف من النص .

◆ fromCharCode ()

هذه الدالة عكس الدالة (charCodAt) حيث إنها ستحول الكود السداسي العشري (أسكي) إلى نص , لاحظ هذا المثال :-

```
var x ;
x = string.fromCharCode ( 65 , 66 , 67 ) ;
```

✚ **ملاحظة //** الحرف A رقمه 65 والحرف Z رقمه 90 والأحرف الأخرى بينهم تزداد بمقدار واحد عن A بالترتيب , أما الحرف a رقمه 97 والحرف z رقمه 122 والأحرف الأخرى بينهم تزداد بمقدار واحد عن الحرف a , ورقم enter هو 13 ورقم esc هو 27 والمسافة (المسطرة) 32 .

◆ الدالة escape و unescape

تعمل الدالة escape على تشفير النص الممرر إليها إلى النظام الستعشري بنظام التشفير(ASCII) مضافاً إليه العلامة % , إما الدالة unescape تعمل على فك تشفير النص الممرر إليها , والصيغة العامة لها هي :-

escape (النص المراد تشفيره) ;

unescape (النص المراد فك تشفيره) ;

مثال /

```
var x = "http://www.4arab.com/index.php?action=" ;
var y = x + escape ( x ) ;
var z = unescape ( y ) ;
alert ( z ) ;
```

◆ الدالة indexOf و lastIndexOf

تستخدم هذه الدوال للبحث عن نص بداخل نص ويوضع بين قوسي هذه الدوال معاملين الأول هو النص الذي تبحث عنه والثاني هو النص الذي تبحث فيه وإذا لم نضع المعامل الثاني وهو يمثل النص الذي نبحث فيه ففي هذه الحالة سوف يبحث في كل النص وهذه الدوال تعود برقم يمثل ترتيب الكلمة في النص .

مثال /

```
var x = " Hello mohamed . How are you Mohamed " ;
var y = x.indexOf ( " Mohamed " ) ;
alert ( y ) ;
```

هذا المثال سيخرج لنا الرقم 24 ولاحظ أنه تجاهل (Mohamed) الأولى وذلك لأنه حساس لحالة الأحرف الكبيرة والصغيرة .

* الفرق بين الدالتين هو أن الدالة indexOf تبدأ البحث من البداية والدالة lastIndexOf تبدأ البحث من النهاية .

◆ الدالة substr و substring

تستخدم هذه الدوال في نسخ جزء من نص , ويكون ناتج الدالتين هو الجزء المنسوخ من النص , حيث أن الدالة substring يتم وضع معاملين بين قوسيهما الأول يمثل رتبة الحرف الأول الذي يبدأ منه النسخ والمعامل الثاني يمثل الحرف الأخير الذي يتوقف عنده النسخ وهو غير مشمول في عملية النسخ , أما الدالة substr فهي أيضاً يوضع بين قوسيهما معاملين الأول يمثل رتبة الحرف الأول الذي يبدأ عنده النسخ والمعامل الثاني يمثل عدد الأحرف التي سوف يتم نسخها .

* في كلا الدالتين إذا لم نضع المعامل الثاني فإنه سوف يتم النسخ من بداية المعامل الأول إلى نهاية النص , مثال /

```
var x = "JavaScript" ;
var y = x.substring ( 0 , 4 ) ;
alert ( y ) ;
```

في هذا المثال ستكون النتيجة كلمة (Java) .

◆ الدالة toUpperCase و toLowerCase

تستخدم الدالة toLowerCase في تغيير حالة الأحرف إلى الصغيرة , وتستخدم الدالة toUpperCase لتغيير حالة الأحرف إلى الكبيرة وتفيد هاتان الدالتان في التخلص من الحساسية لحالة الأحرف عند مقارنة النصوص لبعضها البعض .

مثال /

```
var x = " I Do Not Care About Case " ;
if ( x.toLowerCase ( ) == " i do not care about case " ) {
alert ( " who cares about case ? " ) ; }
```

◆ concat

تعمل هذه الدالة على دمج قيمة نصية بقيمة نصية أخرى , والصيغة العامة لها هي :-

```
" ( النص الثاني ) " . concat ( " النص الأول " ) ;
```

◆ slice

تستخدم هذه الدالة لنسخ جزء من النص بدون التأثير على النص الأصلي , والصيغة العامة لها هي :-

```
" ( موقع نهاية النسخ , موقع بداية النسخ ) " . slice ( " القيمة النصية " ) ;
```

مع ملاحظة إن موقع نهاية النسخ غير مشمول بعملية النسخ , وإذا أردنا أن ننسخ إلى نهاية النص يمكننا أن لا نحدد موقع نهاية النسخ ليقوم بشكل آلي بالنسخ إلى النهاية .

split ◆

تستخدم هذه الدالة لتقطيع النص استناداً إلى جملة التقطيع الممرر لها ثم تقوم بوضع الجملة المقطوعة في مصفوفة ذات بعد واحد , والصيغة العامة لها هي :-

; (النص الذي سيتم التقطيع على أساسه) .split " النص المراد تقطيعه "

او يمكن كتابتها بهذا الشكل :-

; (طول المصفوفة الناتج , النص الذي سيتم التقطيع على أساسه) .split " النص المراد تقطيعه "

مثال /

```
"abcdef".split (" cd " );
```

ينتج عن هذا المثال المصفوفة التالية :-

```
Arr[0] = ab  
Arr[1] = ef
```

مثال /

```
"abcdef".split (" cd " , 1 );
```

ينتج عن هذا المثال المصفوفة التالية :-

```
Arr[0] = ab
```

link -3

وهي تستخدم لتحويل النص إلى رابط تشعبي .

مثال/

```
var x = "this is a hyper link" ;  
x = x.link ("http://www.microsoft.com") ;
```

* هذا الجدول فيه بعض الدوال للتعديل والتنسيق على النصوص , وهي كلها تابعة للكائن String , ولاحظ بأن كل الحروف تكتب صغيرة .

الوصف	الدالة
تعمل التالي ... 	anchor(name)
تعمل التالي < blink>...</ blink>	blink()
تعمل التالي < b>...</ b>	bold()
تعمل التالي <tt>...</tt>	fixed()
تعمل التالي ...	Fontcolor(colorValue)
تعمل التالي ...	Fontsize(size)
تعمل التالي <i>...</i>	Italics()
تعمل التالي ... 	link(url)
تعمل التالي <big>... </big>	big()
تعمل التالي <small>... </small>	small()
تعمل التالي <strick>...</ strick >	strike()
تعمل التالي <sub>...</ sub>	sub()
تعمل التالي <sup> ... </ sup>	sup()

مثال / لأحد هذه الدوال

```
var x=new String ;
x = " The text " ;
var y = x.fontsize( 3 ) ;
```

مثال/

```
x = "My Text";
var y = x.bold ( ) ;
```

ثانياً :- الكائن العددي NUMBER OBJECT

يمكن إنشاء كائن عددي من خلال هذا الكود

```
var x = new Number ( رقم ) ;
```

* ويمكن استخدام خواص الكائن العددي مع المتغير العددي و JavaScript هي سوف تقوم بتغيير المتغير العددي إلى كائن عددي تلقائياً وتجري عليه العملية .

خواص الكائن العددي

fixed

تعمل هذه الدالة على إزالة أرقام معينة بعد العلامة العشرية وتقريب العدد إلى أقرب عددين مثلاً بعد العلامة او ثلاثة أعداد بعد العلامة العشرية .

مثال/

```
var x = 9.056 ;
```

```
var y = x.fixed ( 2 ) ;
```

```
alert ( y ) ;
```

ثالثاً :- كائن التاريخ DATE OBJECT

من أهم استخدامات كائن التاريخ التعرف على التحديثات التي تم إجرائها على الموقع وآخر زيارة قام بها الزائر للموقع , يتم إنشاء كائن التاريخ باستخدام الكود التالي :-

```
var name = new Date ( ) ;
```

ويمكن أن نضع بين قوسيه قيمة تمثل التاريخ والوقت .

* الجدول التالي يوضح أشهر الدوال الموجودة في كائن التاريخ

الوصف	الدالة
تستخدم في الحصول على يوم معين من الشهر وتعود برقم من 1 إلى 31	getDate ()
تستخدم في استخراج الساعة الحالية وتعود برقم من 0 إلى 23	getHours ()
تستخدم في استخراج الدقيقة الحالية وتود برقم من 0 إلى 59	getMinutes ()
تستخدم في استخراج الثانية الحالية وتعود برقم من 0 إلى 59	getSeconds ()
تستخدم في إخراج الملي ثانية الحالية	getMilliseconds ()
تستخدم في الحصول على الوقت الحالي بشكل قيمة نصية	toTimeString ()
تستخدم في معرفة اليوم الحالي من أيام الأسبوع	getdy ()
تستخدم في معرفة الشهر الحالي في السنة وتعود برقم من 0 إلى 11	getMonth ()
تستخدم في معرفة السنة الحالية في شكل قيمة مكونة من أربع قيم	getFolYear ()
تستخدم لإرجاع رقم السنة	getYear ()
تستخدم للحصول على التاريخ الحالي بشكل قيمة نصية	toDateString ()
تستخدم لإرجاع رقم يوم الأسبوع الموجود بالتاريخ وتكون ممثلة لأيام الأسبوع من 0 إلى 6 حيث 0 يعبر عن يوم الأحد	getDay ()
تستخدم لحساب التاريخ بالملي ثانية الذي مر على هذا التاريخ منذ تاريخ منتصف ليل يوم 1 يناير 1970 وترجع قيمة عددية	getTime ()

مثال/ ليعود بتاريخ اليوم

```
var x = new Date ( ) ;
```

```
alert ( x.getDate ( ) ) ;
```

مثال/

```
var x = new Date ( ) ;
```

```
alert ( x.getDay ( ) ) ;
```

مثال/

```
var x = new Date ( ) ;
```

```
alert ( x.getTime ( ) ) ;
```

* في الجدول التالي توجد أهم وظائف التاريخ

الوظيفة	الدالة
تحدد اليوم من الشهر وتكون القيمة المضافة من 1 إلى 31	setDate ()
تحدد الشهر من السنة	setMonth ()
تحدد السنة في شكل أربع أرقام	setFullYear ()
تحدد الساعة	setHours ()
تحدد الدقيقة	setMinutes ()
تحدد الثانية	setSeconds ()
تحدد الملي ثانية	setMilliseconds ()
تستخدم لتحديد قيمة السنوات بالتاريخ	setYear ()

* وعلى ذلك لتغيير السنة إلى سنة 2009 يستخدم الكود التالي :-

myDateObject.setFullYear (2009) ;

مثال/

var x = new Date () ;

x.setDate("2") ;

عملها	الدالة
تستخدم لتحويل التاريخ إلى ما يعادله بتوقيت جرينش	toGMTString
تستخدم لتحويل التاريخ إلى نص على حسب نظام التشغيل الذي يعمل به الجهاز	toLocalString

مثال/

var x = new Date () ;

alert (x.toGMTString ()) ;

◆ **الدالة parse**

تستخدم لتحويل القيمة النصية إلى متغير من نوع تاريخ .

مثال /

```
var x = new Date ( Date.parse ( " wed Jun 16 17:31:01 PDT 2013 " ) );
alert ( a.getFullYear ( ) ) ;
```

رابعاً :- الكائن XMLHttpRequest

هذا الكائن مهم جداً حيث يمكن من خلاله سحب بيانات من السيرفر سراً أي بدون أن يشعر المستخدم بذلك ولإنشاء هذا الكائن نكتب الكود التالي :-

```
var x = new XMLHttpRequest( ) ;
```

لكن لاحظ بأنه في المتصفحات القديمة من الإنترنت أكسبلورر إذا أنشأنا الكائن بنفس الطريقة السابقة فإنها لن تتعرف عليه ويجب إنشائه بهذه الطريقة :-

```
var x = new ActiveXObject( "Microsoft.XMLHTTP" ) ;
```

مثال /

```
<script type="text/javascript">
var ajax = false;
if(window.XMLHttpRequest) {
alert("تم إنشاء الكائن بنجاح");
} else {
alert("متصفحك لا يدعم هذا الكائن");
}
</script>
```

استخدمنا الشرط للتحقق إن كان متصفح الزائر يدعم الكائن او لا , ويمكننا أن نكتب شرط آخر للتحقق إن كان المتصفح هو أكسبلورر او لا والشرط هو :-

```
if(window.ActiveXObject) {
alert ("Ex") ; }
```

خصائص الكائن XMLHttpRequest

◆ الخاصية open

تستخدم هذه الخاصية لسحب ملفات من الخادم وهي تأخذ بارامترين الأول هو نوع السحب هل هو post ام get والبارامتر الثاني هو اسم الملف او مساره ويمكن أن تكون صفحة txt او صفحة php او صفحة اخرى .

ملاحظة/ يفضل أن تستخدم الطريقة get في حالة سحب البيانات من الخادم والطريقة post في حالة إرسال البيانات إلى الخادم .

مثال/

```
var ajax = new XMLHttpRequest();
ajax.open(" get " , "file.txt" );
```

ويمكن أن نرسل للصفحة معلومات وذلك بوضع علامة الأستفهام بعد اسم الصفحة ونضع القيم لترسل بالرابط .

مثال/

```
var ajax = new XMLHttpRequest();
ajax.open(" get " , "page.php?id=10&name=ahmed" );
```

وهنا يمكن أن نأخذ المعلومات الموجودة في الرابط من الصفحة page.php لأنها أرسلت لها وذلك بواسطة الدالة [\$_GET] , لاحظ :-

```
$_GET[' id '];
```

* إذا استخدمنا الطريقة post وأردنا أن نرسل البيانات فسنتكتب البيانات داخل الخاصية send والتي سنتطرق إليها بعد قليل ويجب أن نستخدم الخاصية.setRequestHeader .

مثال/

```
var ajax = new XMLHttpRequest();
ajax.open(" post " , "page.php " );
ajax.setRequestHeader("content-type","application/x-www-form-urlencoded");
ajax.send( "id=10&name=ahmed" );
```

وفي الصفحة page.php سنكتب الكود التالي لأستقبال البيانات لاحظ :-

```
$_POST[' id '];
```


ملاحظة/ هذه الخاصية يمكن أن تحمل باراميتر ثالث وهو يكون اختياري أي يمكن وضعه أو تركه فارغاً وهذا الباراميتر أما أن يأخذ القيمة **true** او **false** والقيمة الافتراضية له هي **true** , إن وضيق هذا الباراميتر هي تحديد عملية المزامنة أي هل يمكن أن تجري عمليات أخرى في الموقع أثناء جلب الصفحة المطلوبة أم لا يجب أن تتوقف كل العمليات إلى أن ينتهي من جلب الصفحة ويفضل دائماً أن تكون القيمة **true** إلا في حالات قليلة عندما يكون هناك ضغط كبير على الخادم والسرعة بطيئة جداً لكن لاحظ بأنه إذا استخدمنا القيمة **true** فيجب أن نستخدم الخاصية **onreadystatechange** وهي تأخذ بشكل اعتيادي الـ **function** المجهول (أي بدون اسم وسنشرحه في المثال العام للخواص) , أما إذا كانت القيمة **false** فيمكن عدم استخدام هذه الخاصية ولا الدالة المجهولة .

```
ajax.open(" get " , "file.txt" , true ) ;
```

◆ الخاصية **onreadystatechange**

تستخدم هذه الخاصية لمراقبة حالة الكائن , عندما تسند هذه الخاصية لفنكشن بالجافا سكربت فسيتم استدعاء تلك الفنكشن في كل مرة تتغير فيها حالة الكائن فمثلاً عند ارسال البيانات تتغير حالة الكائن وكذلك عندما تسحب بيانات تتغير حالة هذا الكائن وبالتالي يتم استدعاء الفنكشن .

◆ الخاصية **readyState**

تستخدم هذه الخاصية لتظهر حالة الارسال او السحب إلى أين وصل وتظهر النتيجة بشكل رقم وكل رقم يشير إلى حالة معينة لاحظ :-

- 0 uninitialized لم تحدث عملية الإتصال بعد
- 1 loading تم الإتصال بالخادم
- 2 loaded تم استقبال الطلب في الخادم
- 3 interactive تجري الآن معالجة الطلب
- 4 complete تم الانتهاء من الطلب والرد جاهز

ويمكن أن تستخدم هذه الخاصية كشرط أي أن يتم التنفيذ إذا أنتها التحميل مثلاً , لاحظ :-

```
var ajax = new XMLHttpRequest();
if( ajax.readyState == 4 ) {
alert "complete" }
```

الخاصية status

هذه الخاصية تبين حالة عملية الإرسال او الأستقبال وإن كان هناك خطأ أم لا وتظهر النتيجة بشكل رقم وكل رقم يشير إلى حالة معينة لاحظ :-

- 200 OK
- 201 Created
- 204 No Content
- 205 Reset Content
- 206 Partial Content
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout
- 411 Length Required
- 413 Requested Entity Too Large
- 414 Requested URL Too Long
- 415 Unsupported Media Type
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported

ويمكن أن تستخدم هذه الخاصية كشرط أي أن يتم التنفيذ إذا لم يكن هناك أي خطأ مثلاً , لاحظ :-

```
var ajax = new XMLHttpRequest();
if( ajax.status == 200 ) {
alert "OK" }
```

الخاصية responseText

تستخدم هذه الخاصية لاستقبال البيانات النصية من الخادم .

الخاصية responseXML

تستخدم هذه الخاصية لاستقبال بيانات XML من الخادم .

الخاصية send

تستخدم هذه الخاصية في نهاية عملية سحب البيانات ونسند لها القيمة null إذا كنا نستخدم الطريقة get ونسند لها قيمة نصية إذا كنا نستخدم الطريقة post وهذه القيمة النصية هي التي سترسل إلى الصفحة المحددة .

مثال/ إذا كانت الطريقة get

```
var ajax = new XMLHttpRequest();
ajax.send(null);
```

مثال/إذا كانت الطريقة post

```
var ajax = new XMLHttpRequest();
ajax.send( "id=10&name=ahmed" );
```

وفي الصفحة الهدف (صفحة الـ php) نستقبل البيانات بواسطة الدالة [\$_POST] أي وكأنا نستقبل البيانات من نموذج إدخال طبيعي , لاحظ :-

```
$_POST[' id '];
```

مثال شامل لكل الخواص السابقة

```
<html dir="rtl">
<head>
<script type="text/javascript">
var ajax = false ;
ajax = new XMLHttpRequest() ;
function grab( data_source , div_id ) {
if( ajax ) {
var ob = document.getElementById( div_id ) ;
ajax.open( " get " , data_source ) ;
ajax.onreadystatechange = function() {
if( ajax.readyState == 4 && ajax.status == 200 ) {
ob.innerHTML = ajax.responseText ;
} }
ajax.send( null ) ;
} }
</head> </script>
<body>
<form>
```

```
<input type="button" onClick="grab('file.txt' , 'left_div')">
</form>
<div id="left_div"> div text </div>
</body>
</html>
```

لاحظ هنا في هذا المثال استخدمنا الـ function الداخلية ولم نعطيها اسم ويسمى هذا النوع من الـ function بالمجهول لأنه لا يعطى اسم وهنا سيتم استدعاء الدالة كلما يحدث تغيير فيها .

الخاصية abort

تستخدم هذه الخاصية لإيقاف العمل (عملية سحب البيانات) .

مثال/

```
var ajax = new XMLHttpRequest();
if( ajax.readyState == 3 ) {
ajax.abort(); }
```

الفصل السابع

(كائنات وأوامر ممنوعة)

◆ جملة with

عندما نقوم باستخدام خصائص properties او وظائف functions تابعة لكائن واحد يمكننا أن نجعل هذا الكائن يشير إلى خصائصه ووظائفه بدون تكرار كتابته قبل الخاصية او الوظيفة , والصيغة العامة لها هي :-

```
{ ( اسم الكائن المستهدف عدم تكراره ) with
الأكواد
ربما تحتوي الأكواد على خصائص او وظائف لهذا الكائن
}
```

مثال / هذا المثال بدون استخدام جملة with

```
<script LANGUAGE="JavaScript">
<!--
var username = "Ahmed" ;
var familyName = window.prompt (" ما هو اسم العائلة ؟ " ) ;
window.alert (" مرحباً بك يا " + username) ;
window.alert (" مرحباً بعائلة " + familyName ) ;
//-->
</script>
```

وكما ترى في المثال قد قمنا بتكرار كتابة الكائن window ويمكننا أن نقوم بإلغاء تكرار كتابة الكائن window وذلك باستخدام جملة with وبين قوسيهما نكتب اسم الكائن window

مثال/ هذا المثال مع استخدام جملة with

```
<script LANGUAGE="JavaScript"><!--
var username = "Ahmed" ;
with ( window ) {
var familyName = prompt (" ما هو اسم العائلة ؟ " ) ;
```

```

alert (" مرحباً بك يا " + username) ;
alert ("مرحباً بعائلة" + familyName ) ; }
//--> </script>

```

ملاحظة// يمكن كتابة اسم الكائن حتى بعد استخدام جملة with لكن هذا يجعل جملة with بدون فائدة .

مثال/

```

with ( window ) {
var familyName = prompt (" ما هو اسم العائلة ؟ " ) ;
window.alert ("مرحباً بعائلة" + familyName ) ; }

```

تداخل جملة with

يسمح بتداخل جملة with كما في المثال التالي :-

```

with ( window ) {
var x = prompt ("ما هو اسمك ؟") ;
with ( document ) {
write ("مرحباً بك يا" + x ) ; }}

```

history

يدل هذا الكائن على تاريخ التصفح والتنقل بين الصفحات المخزنة في متصفح الزائر , وله ثلاث طرق أو أساليب للتحرك ضمن قائمة المحفوظات :-

1. History.go () يفتح عنوان مخزن ضمن قائمة المحفوظات يأخذ عددا موجب أو سالب مثال { History.go (-3) } يوافق النقر على زر Back ثلاث مرات , او يمكن ان نكتب بين قوسيه رابط موقع ليذهب اليه .
2. History.back() يوافق نقر زر العودة Back مرة واحدة .
3. History.forward() يوافق نقر زر التقدم للأمام forward مرة واحدة .

مثال/ ينشأ زرین للتنقل بين العناوين المخزنة في المتصفح .

```
<form>
<input type="button" value="الصفحة التالية" onClick="history.forward();" />
<input type="button" value="الصفحة السابقة" onClick="history.back();" />
</form>
```

مثال/ ينشأ زرین للتنقل بين العناوين المخزنة في المتصفح .

```
<html>
<head>
<script type="text/JavaScript"><!--
function nav ( x ) {
history.go ( x ) ;}
//--> </script>
</head>
<body>
<form>
<input type="button" value="الصفحة السابقة" onClick="nav ( -1 )" />
<input type="button" value="الصفحة التالية" onClick="nav ( 1 )" />
</form>
</body>
</html>
```

مثال/ يحسب عدد الروابط المخزنه

```
alert( history.length ) ;
```

navigator

يستخدم هذا الكائن لفحص واختبار المتصفح وله عدة خواص منها (appName , appVersion , appCodName)

* لاحظ هذا الجدول :-

الكود	معناه
navigator. appName	تقوم بإرجاع اسم المتصفح
navigator. appVersion	تقوم بإرجاع إصدار المتصفح
navigator. appCodName	تقوم بإرجاع الاسم البرمجي

مثال/

```
<html>
<head>
<script type="text/JavaScript"><!--
function browserTest ( ) {
x = navigator. appName ;
y = navigator. appVersion ;
z = navigator. appCodName ;
alert ( " you are using " + x + " version " + y + " which is code named " + z + " .
" ) ; }
//--> </script>
</head>
<body onLoad=" browserTest ( ) ">
<h1> testing all browsers </h1>
</body>
</html>
```


Math Object ◆

وهي كائنات او طرق تقوم بالعمليات الحسابية والصيغة العامة لها هي :-

(الرقم او المتغير) اسم العملية الحسابية . **Math**

مثال / لأخذ الجذر التربيعي للرقم 9

Math.sqrt (9) ;

* لاحظ هذا الجدول فيه أسماء العمليات الحسابية ونكتب كلمة **Math** وبعدها نقطة قبل اسم أي من هذا العمليات .

اسم العملية	الوصف	مثال
abs (x)	القيمة المطلقة ل x	abs (-7.2) = 7.2
round (x)	تقريب x لأقرب عدد حقيقي	round (9.25) = 9
ceil (x)	التقريب لأكبر عدد حقيقي	ceil (9.2) = 10.0 ceil (-9.8) = -9.0
floor (x)	التقريب لأصغر عدد حقيقي	floor (-9.8) = -10.0
PI	تعطي القيمة π (3,14) بسبعة عشر مرتبة بعد الفاصلة	Math.PI ;
E	الثابت e (2,71) بسبعة عشر مرتبة بعد الفاصلة	Math.E ;
cos (x)	جيب تمام ل x	cos (0.0) = 1.0
sin (x)	جيب x	sin (0.0) = 0.0
tan (x)	ظل x	tan (0.0) = 0.0
sqrt (x)	الجذر التربيعي لx	sqrt (900.0) = 30.0
pow (x , y)	x مرفوعة للأس y (x ^y)	pow (2.0 , 7,0) = 128.0
exp (x)	طريقة الأس (e ^x)	exp (1.0) = 2.71828
log (x)	لوغارتم (x)	log (2.71828) = 1.0
max (x , y)	مقارنة x مع y واستخراج الأكبر	max (2 , 3.5) = 3.5
min (x , y)	مقارنة x مع y واستخراج الأصغر	max (2 , 3.5) = 2
random ()	تولد أرقام عشوية بشكل عشوائي بين الصفر والواحد الصحيح وهذه القيم لا تحتوي على الواحد ولا على الصفر	random () = 0.1

* أهم هذه العمليات الحاسوبية هو random والذي يمكن من خلاله توليد أرقام عشوائية كبيرة وذلك بضربه برقم آخر وللتخلص من الرقم العشري ونكون رقم صحيح (بدون فاصلة) نستخدم معه العملية floor , حيث يمكن أن نستفيد من هذا الرقم العشوائي في أمور كثيرة .

مثال / لإظهار نص جديد كلما فتح المستخدم الصفحة

```
<script LANGUAGE="JavaScript"><!--
var myArray = new Array ( ) ;
myArray[0] = " Hello " ;
myArray[1] = " Hi " ;
myArray[2] = " Welcome " ;
var i = Math.random ( ) * 3 ;
i = Math.floor ( i ) ;
document.write ( myArray[i] ) ;
//--> </script>
```

◆ التايمر setTimeout

يستخدم التايمر (المؤقت) لتنفيذ وظيفة ما بعد مدة نحن نحددها له بالملي ثانية (كل 1000 ملي ثانية يساوي 1 ثانية) والصيغة العامة له هي :-

```
setTimeout ( "الوظيفة التي ستنفذ" , الزمن بالملي ثانية , "الوظيفة التي ستنفذ" ) ;
```

مثال/

```
<body onload="mofak ( )" >
<script LANGUAGE="JavaScript"><!--
function hh ( ) {
alert ("welcome in my site"); }
function mofak ( ) {
setTimeout ( hh() , 3000 ) ; }
//--> </script>
</body>
```

حيث استدعينا الدالة mofak عند تحميل الصفحة والتي تحتوي بداخلها على التايمر والذي يشير بداخله إلى الدالة hh وهي التي تحمل الوظيفة التي ستنفذ .

مثال/ لعمل عداد يستمر بالعد إلى مالا نهاية

```
<body onload="mofak ()">
<form name="f1">
<input type="text" name="t1" />
</form>
<script LANGUAGE="JavaScript"><!--
var i = 0 ;
function mofak () {
i ++ ;
fl.t1.value = i ;
setTimeout ( mofak () , 100 ) ; }
//--> </script>
</body>
```

حيث في البداية حملنا الدالة mofak ثم جعلنا الدالة mofak تستدعي نفسها من داخلها في داخل setTimeout وهذه الخدعة جعلت الدالة (العداد) تدور إلى مالا نهاية , ولعمل عدد دورات محدد نضع التايمر setTimeout في داخل أداة شرط ونشروط عند وصول العداد i إلى عدد معين يتوقف ولا ينفذ ما بداخل أداة الشرط والتي تحتوي في داخلها التايمر setTimeout . ويمكننا أن نجعل العداد يعرض في داخل وسم <div> وبذلك نستطيع أن نتحكم بلون وحجم العداد بشكل أفضل , ويمكننا عمل تايمر بحيث يقوم بتغيير لون الصفحة من وقت لآخر ويمكن إسناد قيم لونية وجعلها تتغير بشكل تدريجي من خلال رقم درجة الإشباع للون .

clearTimeout

يستخدم هذا المنهج لإيقاف التايمر (setTimeout) بمعنى انه سيمسح المؤقت .

مثال/

```
<body onload="mofak ()">
<script LANGUAGE="JavaScript"><!--
function hh () {
alert ("welcome in my site");
clearTimeout ( x ) ;
}
function mofak () {
setTimeout ("hh" , 3000) ; }
//--> </script>
</body>
```

setInterval ◆

هذه الدالة هي تماماً مثل الدالة setTimeout لكن الفرق الوحيد بينهما هو ان هذه الدالة تنفذ الكود بشكل متكرر بخلاف الدالة setTimeout التي تنفذ الكود مرة واحدة بعد مرور وقت محدد
مثال/

```
function hh ( ) {
alert ("welcome in my site"); }
setInterval ("hh" , 3000) ;
```

clearInterval ◆

هذه الدالة تحذف الدالة السابقة setInterval
مثال/ لتنفيذ الكود اربع مرات فقط ومن ثم يتوقف (يحذف الدالة setInterval)

```
var i = 1 ;
function hh ( ) {
alert ("welcome in my site");
if( i >= 4 ) {
clearInterval( s ) ;
i += 1 ; }
var s = setInterval ("hh" , 3000) ;
```

Shell ◆

هذه الدالة تعمل فقط مع المتصفح إكسبلورر , تستخدم هذه الدالة لفتح النوافذ Window الخاصة بحاسوب المستخدم .

مثال / لفتح برنامج المفكرة Notepad

```
<body>
<input type="button" value="open" onclick="TakeNotes();">
<script language="javascript">
function TakeNotes ( ) {
var o = new ActiveXObject ("WScript.Shell");
o.Run ("NOTEPAD.exe") ;
o = null ; }
</script>
</body>
```

* ويمكن كذلك فتح المكتبة Libraries وذلك فقط باستبدال كلمة ("NOTEPAD.exe") بكلمة ("explorer.exe") , ويمكن كذلك عمل إغلاق للجهاز وذلك فقط باستبدال كلمة ("NOTEPAD.exe") بكلمة ("Shutdown /s") , ويمكن أيضاً عمل إعادة تشغيل للجهاز وذلك فقط باستبدال كلمة ("NOTEPAD.exe") بكلمة ("Shutdwon /r")

الفصل الثامن (تجنب الأخطاء والأنماط)

أولاً :- تجنب وقوع الأخطاء

يمكن تجنب وقوع الأخطاء باستخدام صيغة try و catch , والصيغة العامة لها هي :-

```
try {
    الكود } catch ( e ) {
    الكود عند الخطأ }
```

حيث إنه سيتم كتابة الأكواد بين أقواس الجملة try فإذا حدث أي خطأ ينتقل المفسر إلى منطقة الأكواد بين أقواس الجملة catch ويتم تحميل كائن مخصص لحمل مواصفات هذا الخطأ جرت العادة لتسمية هذا المتغير بالاسم e لأنه يدل على حدث استثناء Exception

مثال/

```
<script LANGUAGE="JavaScript">
<!--
var num1 , num2 ;
try {
    alert ( num3 ) ;
    } catch ( e ) {
    alert ( " حدث خطأ " ) ; }
//-->
</script>
```

يمكن إظهار رسالة الخطأ الأساسية باستخدام خصائص الكائن e وهذه الخصائص هي :-

- | | |
|----------------|--------------------------------|
| 1- name | تعبر عن اسم الخطأ . |
| 2- message | تعبر عن محتوى نص رسالة الخطأ . |
| 3- number | تعبر عن رقم الخطأ . |
| 4- description | تعبر عن رسالة تفصيلية للخطأ . |

مثال/

```

<script LANGUAGE="JavaScript">
<!--
var num1 , num2 ;
try {
    alert ( num3 ) ;
} catch ( e ) {
    alert ( " اسم الخطأ : " + e.name + "\n" +
        " رقم الخطأ : " + e.number + "\n" +
        " رسالة الخطأ : " + e.message + "\n" +
        " وصف الخطأ : " + e.decription ) ;
}
//-->
</script>

```

استحداث الأخطاء

نحتاج في بض الأحيان إلى استحداث خطأ ما ويتم ذلك باستخدام الأمر `throw` , وذلك بطريقتين :-

الطريقة الأولى :-

لاحظ هذا المثال ليس فيه خطأ لكننا سوف نستحدث فيه خطأ

```

<script LANGUAGE="JavaScript">
<!--
var x = "مرحباً بك";
try {
    alert ( x ) ;
    throw " هذا خطأ غير حقيقي ولكنه مستحدث " ; } catch {
    alert ( " - رسالة الخطأ : " + e ) ; }
//-->
</script>

```

ستظهر الرسالة التالية (رسالة الخطأ :- هذا خطأ غير حقيقي ولكنه مستحدث) , حيث انه يتم كتابة رسالة الخطأ بعد الأمر `throw` وسينتقل مباشرة إلى الجملة `catch` حاملاً معه الخطأ في المتغير `e` .

الطريقة الثانية :-

عمل خطأ من الكائن `Error` ثم إرساله إلى الأمر `throw`

مثال /

```

<script LANGUAGE="JavaScript">
<!--
var x ;
try {alert ( x ) ;
var err = new Error ( " هذا خطأ غير حقيقي ولكنه مستحدث " ) ;
err.description = " هذا خطأ تم صنعه يدوياً " ;
throw err ; } catch ( e ) {
alert ( " رسالة الخطأ : " + e.message + "\n" + " وصف الخطأ : " + e.decription ) ;
}
//-->
</script>

```

ثانياً :- الأنماط patterns

وهي تستخدم لعمليات البحث والاستبدال وهي تكتب بين علامات سلاش / / .

مثال /

```
var patren = / hello / ;
```

فهذا النمط يعبر عن الكلمة hello ويمكن أن نضع في النهاية الحرف (i) حيث إن هذا يعني أن يتم البحث عن هذه الكلمة سواء كانت حروفها صغيرة او كبيرة او قسم منها كبير وقسم صغير , او يمكن أن نضع في الأخير الحرف (g) ليتم البحث عن هذه الكلمة إلى نهاية النص بدون توقف , ويمكن الجمع بين (i) و (g) .

مثال /

```
var pattern = / web / i ;
```


* ويمكن إضافة مجموعة من الأحرف والرموز لنتمكن من البحث بشكل أدق عما نريد , لاحظ هذا الجدول :-

الوصف	الرمز
تعبّر عن حرف واحد (ياً كان هذا الحرف) .	.
تعبّر عن وجود حرف واحد او عن عدم وجود أي حرف .	?
تعبّر عن وجود حرف واحد او أكثر او لا شيء .	*
تعبّر عن وجود حرف واحد او أكثر .	+

مثال /

`var pattern = /c.t/ ;`

هذا المثال يعبر عن كلمة تبدأ بالحرف c وتنتهي بالحرف t وبينهم حرف معين (أياً كان هذا الحرف)

مثال/

`var pattern = /w?eb/ ;`

هذا المثال يعبر عن كلمة web او كلمة يقع بين حرف w والحرف e حرف آخر مثلاً قد تكون الكلمة weeb او wzeb او أي كلمة أخرى .

* هذا الجدول يوضح رموز البداية والنهاية الخاصة

الوصف	الرمز
تعبّر عن إنه عند إجراء عملية تماثل النص مع النمط يجب أن يتكافأ مع بداية النص .	^
تعبّر عن إنه عند إجراء عملية تماثل النص مع النمط يجب أن يتكافأ مع نهاية النص .	\$

مثال/

`var pattern = /^web/ ;`

في هذا المثال اشترطنا أن يبدأ النص بكلمة web .

* هذا الجدول يوضح رموز التكرار العديدة

الوصف	الرمز
تعبّر عن إنه يجب تكرار الحرف السابق له عدد من المرات يساوي n .	{ n }
تعبّر عن إنه يجب تكرار الحرف السابق له عدد من المرات يساوي من القيمة n إلى القيمة m .	{ n , m }
تعبّر عن إنه يجب تكرار الحرف السابق له عدد من المرات يساوي من القيمة n إلى ما لانهاية .	{ n , }

مثال /

```
var pattern = /c { 2 } t / ;
```

هذا النمط يمكن أن يعبر عن " cct "

* هذا الجدول يوضح رموز المدى

الوصف	الرمز
تعبير عن إنه يتم البحث في المدى المحدد	[]
تعبير عن إنه يتم البحث في عكس المدى المحدد	[^]

مثال / هذا النمط يعبر عن أي حرف صغير

```
var pattern = / [ a-z ] / ;
```

مثال/ هذا النمط لا يعبر عن أي حرف صغير

```
var pattern = / [ ^ a-z ] / ;
```

الشرح	النمط
يعبر عن جميع الحروف صغيرة وكبيرة .	[a-zA-Z]
لا يعبر عن جميع الحروف صغيرة وكبيرة .	[^ a-zA-Z]
يعبر عن جميع الأرقام من صفر إلى 9 .	[0-9]
لا يعبر عن جميع الأرقام من صفر إلى 9 .	[^ 0-9]
يعبر عن جميع الأعداد والحروف كبيرة وصغيرة .	[a-zA-Z0-9]
لا يعبر عن جميع الأعداد ولا الحروف كبيرة ولا صغيرة .	[^ a-zA-Z0-9]

* جدول يوضح استخدام الرموز الخاصة

الوصف	الرمز
يعبر عن الأرقام من صفر إلى 9 أي أنه يكافئ النمط [0-9]	\d
لا يعبر عن الأرقام من صفر إلى 9 أي أنه يكافئ النمط [^0-9]	\D
تعبر عن جميع الرموز التي يمكن كتابتها بالأرقام من صفر إلى 9 او عن الحروف كبيرة او صغيرة أي أنه يكافئ النمط [a-zA-Z0-9] .	\w
لا تعبر عن الرموز التي يمكن كتابتها بالأرقام من صفر إلى 9 ولا عن الحروف كبيرة او صغيرة أي أنه يكافئ النمط [^a-zA-Z0-9] .	\W
تعبر عن جميع الرموز الخاصة والتي يصعب كتابة بعضها مثل حرف المسافة و التاب tab والسطر الجديد \n والرموز التالية \r و \f .	\s
هو عكس الرمز السابق .	\S
تعبر عن أن النمط يجب أن يتواجد في أول كلمة فقط على سبيل المثال النمط / \bor/ يعبر عن organ ولا يعبر عن normal والنمط /or\b/ يعبر عن traitor ولا يعبر عن perform .	\b
هو عكس الرمز السابق .	\B

الوصف	الرمز
تعبر عن الحرف \	\\
تعبر عن الحرف tab	\t
تعبر عن الحرف .	\.
تعبر عن الحرف ?	\?
تعبر عن الحرف +	\+
تعبر عن الحرف *	*
تعبر عن الحرف ^	\^
تعبر عن الحرف \$	\\$

* ولعمل مجموعة من هذه الرموز السابقة (الأنماط) يتم وضعها بين قوسين () , مثال /

```
var pattern = /^[ 0-9 ] (web) / ;
```

* ويمكن إجراء عملية منطقية على النصوص باستخدام العلامة (|) وهي مثل كلمة (او)

مثال /

```
var pattern = /web | cat/ ;
```

حيث يمكن أن يعبر هذا النمط عن cat او web .

استخدام كائن النصوص مع الأنماط

match

تستخدم هذه الدالة للبحث عن نمط معين بداخل النص ثم تقوم بإرجاع مصفوفة بها نواتج البحث و إذا لم يكن النمط موجود في النص تعود بالقيمة null , والصيغة العامة لها هي :-

; (النمط) .match النص

مثال /للتأكد من صحة البريد الالكتروني الذي يتم استقباله من المستخدم

```
<script LANGUAGE="JavaScript"><!--
var email = prompt ( " أدخل بريدك الالكتروني " , " " ) ;
var emailCod = /^ [ a-zA-Z ] {1} \w + @ [ a-zA-Z ] {1} \w + \. \w {2 , 3} / ;
var arr = email.match ( emailCod ) ;
if ( arr != null ) { alert ( " صح " ) ;
} else { alert ( " خطأ " ) ;
}
//--> </script>
```

replace

تستخدم هذه الدالة للبحث عن نمط معين بداخل النص ثم تقوم باستبداله بنص آخر , والصيغة العامة لها هي :-

; (النص الجديد , النمط) .replace " النص "

مثال /

```
function repp ( ) {
var r , re ;
var s = "the quick fox over laze" ;
re = /fox/ ;
r = s.replace ( re , "pig" ) ;
document.write ( r ) ;
}
```

search

تستخدم هذه الدالة للبحث عن نمط معين بداخل النص ثم تقوم بإرجاع مكان تواجده (موقع النص على هيئة قيمة عددية) مثل الدالة indexOf التابعة للكائن النصي , وإذا لم يجد الكلمة داخل النص ستعيد القيمة 1- , والصيغة العامة لها هي :-

; (النمط) .search " النص "

مثال/

```
var x="how are you ?";  
var y=x.search("you");  
document.write( y ) ;
```

كائن التعبيرات المنتظمة REGEXP

يمكننا إنشاء كائن التعبيرات المنتظمة RegExp كما يلي :-

```
var x = new RegExp ( ) ;
```

خصائص الكائن RegExp

1- الخاصية global

هذه الخاصية للقراءة فقط وتعبر هل النمط المستخدم تم استخدام المعامل g به , فعلى سبيل المثال النمط التالي / hello /g تكون الخاصية global له تساوي true أما النمط التالي / hello / تكون الخاصية global له تساوي false .

2- الخاصية ignoreCase

هذه الخاصية للقراءة فقط وتعبر هل النمط المستخدم تم استخدام المعامل i به , فعلى سبيل المثال النمط التالي / hello /i تكون الخاصية ignoreCase له تساوي true أما النمط التالي / hello / تكون الخاصية ignoreCase له تساوي false .

3- الخاصية multiline

هذه الخاصية للقراءة فقط وتعبر هل النمط المستخدم سوف لا يتجاهل السطور العديدة به .

4- الخاصية source

هذه الخاصية للقراءة فقط , وتعبر عن صيغة النمط المستخدم بالكائن RegExp .

دوال الكائن REGEXP

● الدالة compile

تمكننا من إنشاء كائن للتعبيرات المنتظمة به نمط معين , والصيغة العامة لها هي :-

RegExp.compile (النمط) ;

او يمكن استخدام هذه الصيغة

RegExp.compile (النمط , ["g" | "i" | "m"]) ;

حيث أن هذه الصيغة الثانية تحدد طريقة البحث باستخدام النمط , حيث إن الرمز m يعبر عن البحث في جميع السطور .

مثال/

```
var x = new RegExp ( ) ;
```

```
var y = x.compile ("/me/");
```

```
var y = x.compile ("/me/", " i " );
```

◆ الدالة test

تستخدم للتأكد من توافق النمط مع النص الممرر لها فإذا كان النمط متوافق مع النص ترجع القيمة true وفي حال لم يكن النمط متوافقاً مع النص ترجع القيمة false , والصيغة العامة لها هي :-

RegExp.test (النص) ;

مثال /

```
var RegExp = /me/ ;
```

```
alert ( RegExp.test ( " hello me hello " ) ) ;
```

في هذا المثال ستعيد القيمة true

◆ الدالة exec

تستخدم هذه الدالة لتنفيذ النمط على النص الممرر لها وسوف تعيد لنا مصفوفة , والصيغة العامة لها هي :-

RegExp.exec (النص) ;

مثال /

```
var RegExp = /me/ ;
```

```
var x = RegExp.exec ( " hello me hello " ) ;
```

```
if ( x == null ) {
```

```
  alert ( " لم يتم العثور على كلمة me " ) ;
```

```
} else { alert ( " تم العثور على كلمة me " + x[0] ) ; }
```

الفصل التاسع

(النماذج والإطارات)

أولاً :- النماذج

يمكننا التحكم بالنماذج من خلال الأمر forms والذي هو أحد أوامر الكائن document وكما نعرف فإنه يتم الفصل بين الأمر والكائن بنقطة , والنموذج من وجهة نظر JavaScript يعتبر كائن مصفوفة ثنائية الأبعاد والحقول التي يحتوي عليها النموذج تعتبر عناصر هذه المصفوفة الرئيسية وهي مصفوفات فرعية , وبهذه الحالة يمكن استخدام الدوال الخاصة بالمصفوفات مع النماذج , إذا أردنا أن نجري عملية او نطبق تأثير ما على كل النماذج (كل وسوم <form>) الموجودة في الصفحة سنكتب هذا الكود

; اسم الدالة او الوظيفة التي سنطبقها على النموذج .document.forms

مثال/

document.forms.length ;

في هذا المثال سوف يحسب كم نموذج إدخال (وسم <form>) موجود في الصفحة ويمكن أن نجري أي عملية أخرى , لكن إذا أردنا أن نجري عملية ما على نموذج معين فيمكننا الوصول إلى ذلك النموذج من خلال كتابة رقم تسلسله في الصفحة يعني هل هو أول وسم <form> أم ثاني وسم أو أي وسم ويتم الحساب من أعلى الصفحة (أول الصفحة) إلى أسفل الصفحة (آخر الصفحة) , مثلاً إذا أردنا تحديد الوسم <form> الثاني فسنكتب الرقم 1 بين قوسي الأمر forms هكذا forms[1] طبعاً كتبنا الرقم 1 لأننا كما قلنا النماذج تعتبر كمصفوفة والمصفوفة تبدأ بالرقم 0 ثم 1 ثم 2 وهكذا , كما نعرف فإن الوسم <form> يحتوي بداخله على الوسم <input /> فإذا أردنا أن نتعامل مع الوسم <input /> سنحدد أولاً الوسم <form> الحاوي له ثم نحدد الوسم <input /> من خلال الأمر elements وبعد ذلك سنكتب الوظيفة او الدالة التي نريد تطبيقها على الوسم <input />

مثال/

document.forms[0].elements.length ;

وفي هذا المثال سيحسب كم وسم `<input />` موجود في داخل أول وسم `<form>` , وإذا أردنا تحديد وسم `<input />` معين في داخل هذا النموذج فيجب أن نكتب رقم تسلسل الـ وسم `<input/>` الموجود في داخل هذا النموذج بين قوسى الأمر `elements` وكما فعلنا مع الـ وسم `<form>` مثلاً إذا أردنا تحديد ثاني وسم `<input />` في النموذج فسنكتب `elements[1]` وكما قلنا سابقاً يبدأ العد في المصفوفات من الرقم 0 .

مثال /

```
document.forms[0].elements[0].value="hello" ;
```

value هي خاصية تستخدم للتحكم بالقيمة الموجودة في داخل الحقل النصي او الزر button الموجود في النموذج (أي أنها تساوي الخاصية value التي تكتب بداخل الـ وسم `<input />`) وفي هذا المثال ستظهر الكلمة hello في داخل المربع النصي الذي حددناه .

إذا كان لدينا مجموعة كبيرة من النماذج وفيها حقول إدخال وأزرار كثيرة سيكون من الصعب تحديد رقم تسلسل كل منها لذلك يمكننا أن نستغل الخاصية name والتي هي من خصائص الـ وسم `<form>` وكذلك من خصائص الـ وسم `<input />` وهنا سنذكر فقط الاسم الذي نعطيه للخاصية name , حيث سنذكر اسم الكائن document ثم اسم الـ وسم `<form>` ثم اسم الـ وسم `<input/>` ثم القيمة او الدالة وكما تعرف بفصل بين كل منهم بنقطة , إذا يمكن كتابة نفس المثال السابق بهذا الشكل :-

```
document.fo.in.value="hello" ;
```

اسم الـ وسم `<input />`
 اسم الـ وسم `<form>`

مثال/ لاختبار الحقول التي يدخلها المستخدم إن كانت مملوءة او لا

```
<html> <head>
<script LANGUAGE="JavaScript">
<!--
function testForm ( ) {
if ( documrnt.ff.x.value = " " || documrnt.ff.y.value = " " ) {
alert ( " يجب الكتابة في كل الحقول " );}
}
//-->
</head> <body>
<form name="ff" onSubmit="testform ( ) ;" >
<input type="text" name="x" />
```

```

<input type="text" name="y" />
<input type="submit" />
</form>
</body> </html>

```

ملاحظة // يمكن أن لا نكتب اسم الكائن document قبل اسم النموذج <form> أي أن نكتب مباشرة اسم النموذج وبعده اسم حقل الإدخال وبعده اسم الدالة أو القيمة , أي إننا مخيرون في كتابة أو عدم كتابة اسم الكائن document , لكن هذا الملاحظة تصح فقط إذا كنا قد أعطينا اسم للوسم <form> واسم للوسم </input /> وكنا نتعامل من خلال هذه الأسماء أما إذا كنا نستخدم الأوامر الافتراضية وهي forms و elements فهنا يجب كتابة اسم الكائن الرئيسي وهو document .

مثال/ يمكن أن يكون شكل الدالة function في المثال السابق بهذا الشكل

```

function testForm ( ) {
if ( ff.x.value == " " || ff.y.value == " " ) {
alert ( " يجب الكتابة في كل الحقول " );
}
}

```

مثال / لجمع رقمين نأخذهم من المستخدم

```

<html>
<head>
<script LANGUAGE="JavaScript">
<!--
function plus ( ) {
var z ;
var x = document.ff.a.value ;
var y = document.ff.b.value ;
z = Number(x) + Number(y) ;
document.ff.c.value = z ;
}
//-->
</script>
</head>
<body>
<form name="ff">
الرقم الأول : <input type="text" name="a" />
الرقم الثاني : <input type="text" name="b" />
<br />
النتج : <input type="text" name="c" />

```

```
<input type="button" onclick="plus()" />
</form>
</body>
</html>
```

◆ الغرض this

لهذا الغرض عدة مناهج وأحد هذه المناهج هو المنهج form وكما نعرف نفصل بين الغرض والمنهج بنقطة , حيث إذا كتبنا هذا المنهج في نموذج إدخال معين فإنه يشير إلى هذا الحقل الموجود به هذا المنهج , ولتوضيح الفكرة نأخذ نفس المثال السابق (لجمع رقمين نأخذهم من المستخدم) لكن بطريقة أخرى مع إن النتيجة واحدة .

مثال /

```
<html>
<head>
<script LANGUAGE="JavaScript"><!--
function plus ( feed ) {
var z ;
var x = feed.a.value ;
var y = feed.b.value ;
z = Number(x) + Number(y) ;
feed.c.value = z ; }
//--> </script>
</head>
<body>
<form>
الرقم الأول : <input type="text" name="a" />
الرقم الثاني : <input type="text" name="b" />
<br />
النتائج : <input type="text" name="c" />
<input type="button" onclick="plus( this.form )" />
</form>
</body>
</html>
```

كما لاحظت فقد كتبنا الغرض this.form في داخل قوسي الدالة عند الاستدعاء وهو بذلك سيعيد القيمة إلى الباراميتير feed الذي وضعناه في الدالة function وكما تعرف يمكن وضع أي اسم كباراميتير داخل الدالة function وهذا الباراميتير يكتب بدلاً من اسم الكائن document وبدلاً من اسم النموذج <form> وبعده نضع نقطة وبعدها نكتب اسم الحقل <input />

ملاحظة // مع الغرض this لا يمكن أن نكتب اسم الكائن الأب document كما لاحظت في المثال السابق وإذا كتبناه سيعتبر خطأ .

مثال / بدون استخدام this

```
<form name="ff">
<input type="button" name="bb"
onmouseover="document.ff.bb.value=' Mouse Over ' ; " />
</form>
```

مثال/ نفس المثال السابق لكن مع استخدام this

```
<form name="ff">
<input type="button" name="bb"
onmouseover="this.value=' Mouse Over ' ; " />
</form>
```

* كما ذكرت في بداية شرح النماذج بأن JavaScript تعتبر النموذج كمصفوفة وبذلك يمكن أن نستخدم دوال المصفوفات معها حيث سنكتب الدوال بدلاً من كلمة (value) التي تشير إلى القيمة , مثلاً يمكننا أن نكتب الدالة (length) لمعرفة عدد الحقول الموجودة والتي تعود لنفس النوع .

مثال/

```
<html>
<head>
<script LANGUAGE="JavaScript"> <!--
function plus ( ) {
alert ( ff.list.length ) ; }
//--> </script>
</head>
<body>
<form name="ff">
<input type="checkbox" name="list" />
<input type="checkbox" name="list" />
<input type="checkbox" name="list" />
<input type="button" onclick="plus()" />
</form>
</body>
</html>
```

ويمكن كتابة نفس المثال لكن باستخدام الغرض this.form لكن هنا سنكتب بعده اسم الحقول أيضاً .

المثال /

```

<html>
<head>
<script LANGUAGE="JavaScript"><!--
function plus ( feed ) {
alert ( feed.length ) ;}
//--> </script>
</head>
<body>
<form>
<input type="checkbox" name="list" />
<input type="checkbox" name="list" />
<input type="checkbox" name="list" />
<input type="button" onclick="plus(this.form.list)" />
</form>
</body>
</html>

```

focus و select ◆

يستخدم select لتحديد (تظليل) نص معين , أما focus فهي تستخدم لتحويل مؤشر إلى مكان معين .

مثال /

```

<html>
<head>
<script type="text/JavaScript"><!--
function sAll ( ss ) {
ss.focus ( ) ;
ss.select ( ) ; }
//--> </script>
</head>
<body>
<form name=" f1">
<input type="text" name="t" value="the text" />
<input type="button" onclick=" sAll ( document.f1.t ) ; " />
</form>
</body>
</html>

```

مثال/ لاستخدام this مع select لتحديد نص معين

```
<html>
<head> </head>
<body>
<form>
<input type="text" value="FTI" onclick="this.select ( )" />
</form>
</body>
</html>
```

1- الأزرار BUTTONS

تعتبر الأزرار أكثر عناصر النموذج استخداماً , يتم إنشاء الزر في لغة html باستخدام هذا الوسم :-

```
<input type="button" name="mybutton" value="اضغط هنا" />
```

أحداث الزر Button

أ- onClick

تستخدم هذه الدالة لتحديد ما يحدث عند قيام الزائر بالضغط على زر النموذج .

مثال/

```
<input type="button" onClick="namefunction ( )" />
```

حيث أن namefunction هو اسم الدالة التي سيتم تنفيذها عند الضغط على الزر .

ب- onMouseup و onMousedown

يستخدم الحدث onMousedown لتنفيذ الأمر عند قيام المستخدم بالضغط فوق زر النموذج , والحدث onMouseup عندما يترك المستخدم زر الماوس .

مثال/

```
<input type="button" onmousedown="functiondown()"
onmouseup="functionup()" />
```

حيث أن functionup و functiondown تشير إلى أسماء دوال تم تعريفها مسبقاً ليتم تنفيذها

2- الحقل النصي

ما يتم إدخاله في الحقل النصي لا يمكننا إجراء العمليات الحسابية عليه لأن JavaScript ستعامله كأنه نص حتى لو أدخل المستخدم فيه قيمة عددية , مثلاً لو أدخل المستخدم الرقم 1 وحاولنا الجمع مع 1 ستكون النتيجة 11 .

وللحقل النصي مجموعة من الأحداث كما تحدثنا سابقاً عن أحداث الزر , ومن أحداث الحقل النصي التالي :-

- 1- onselect
- 2- onkeydown
- 3- onkeypress
- 4- onkeyup
- 5- onfocus
- 6- onblur

مثال/ لتغيير حجم الحقل النصي

```
<form name="ff">
<input type="text" name="tt" size="30" />
<input type="button" onclick="ff.tt.size='10' ; " />
</form>
```

3- مربعات وأزرار الاختيار

ويستخدم معها مجموعة من الأحداث :-

onclick -1

onfocus -2

onblur -3

* يمكننا معرفة أي من أزرار الاختيار تم تحديده (مفعّل) وذلك بكتابة checked بدلا من القيمة value التي تكتب بعد اسم النموذج وحقل الإدخال المستخدمة لتحديد النموذج , لاحظ هذا المثال لتتضح الفكرة .

مثال /

```
<html>
<head>
<script LANGUAGE="JavaScript">
function plus ( ) {
if ( ff.col1[0].checked ) {
alert ( "انت اخترت لا وهي ايضا لا تحبك" ) ; }
if ( ff.col1 [1].checked ) {
alert ( "انت اخترت نعم وهي ايضا تحبك" ) ; }
}
</script>
</head>
<body>
<b> هل تحب لغة جافا سكربت ؟ </b>
<form name="ff">
<p><input type="radio" name="col1" value="a" /> لا </p>
<p><input type="radio" name="col1" value="b" /> نعم </p>
<br />
<input type="button" onclick="plus()" value="Example" />
</form>
</body>
</html>
```

لاحظ أننا عرفنا أن الأداة مفعّلة أم لا من خلال (ff.col1(0).checked) , ولاحظ أننا كتبنا الرقم 0 وهذا يعني الأداة ذات الترتيب الأول في الصفحة والأداة الثانية تحمل الرقم 1 وهكذا , ولاحظ أن الأدوات كلها تابعة لنفس المجموعة وهي col1 .

مثال/ لإنشاء زر يفعل كل خانات الاختيار وزر يلغي تفعيل كل الخانات

```
<html>
<head>
<script LANGUAGE="JavaScript">
function check ( feed ) {
for ( i=0;i<feed.length;i++ ) {
feed[i].checked = 1 ;
} }
function discheck ( feed ) {
for ( i=0;i<feed.length;i++ ) {
feed[i].checked = 0 ;
} }
</script>
</head>
<body>
<form>
<input type="checkbox" name="list" /> <br />
<input type="checkbox" name="list" /> <br />
<input type="checkbox" name="list" /> <br />
<input type="checkbox" name="list" /> <br />
<input type="button" onclick="check(this.form.list)" value=" تحديد الكل " />
<input type="button" onclick="discheck(this.form.list)" value="إلغاء تحديد الكل" />
</form>
</body>
</html>
```

* القيمة checked تحدد إذا كان العنصر مفعّل أم لا حيث إذا كانت تحمل القيمة 1 أو true يكون العنصر مفعّل وإذا كانت 0 أو false يكون العنصر غير مفعّل , والدالة length تستخدم لتحديد عدد صناديق الاختيار .

* القيمة () click أيضاً تحدد إذا كان العنصر مفعّلاً أم لا وهي لا تأخذ أي قيمة بل نكتها مباشرةً , مثلاً بالنسبة للمثال السابق يمكن أن نحذف القيمة checked هي وقيمتها ونضع بدلها القيمة () click بدون أي قيمة (نترك قوسيه فارغين) . مع ملاحظة أنه يمكن أن نستخدم أي من هذه القيم مع عنصر الإدخال radio كما فعلنا مع عنصر الإدخال checkbox .

4- قوائم الاختيار

يمكننا معرفة كم عنصر في القائمة من خلال استخدام الخاصية length الخاصة بكائن القائمة , حيث سنتعامل مع قائمة الاختيار على اعتبار أنها مصفوفة , ويمكن أيضاً إضافة عنصر للقائمة وذلك بإضافة عنصر إلى المصفوفة option وهي تمثل القائمة , كما في الكود التالي :-

```
var myNewOption=new option ( "the text" , "the value" ) ;
```

ويمكننا أيضاً إزالة عنصر من القائمة كما في الكود التالي :-

```
document.theForme.theSelectOption[0] = null ;
```

مثال / لإضافة الأعوام من 1950 إلى عام 2020 كلها في قائمة اختيار

```
<script LANGUAGE="JavaScript">
document.write(" <form> <select size='1'> " ) ;
for ( i=1950;i<=2020;i++ ) {
document.write(" <option> " ) ;
document.write(i) ;
document.write(" </option> " ) ;
}
document.write("</select> </form>") ;
</script>
```

الخصائص التي يمكن استخدامها للتأثير على النماذج

* هذا الجدول يوضح الخصائص التي يمكن كتابتها بعد اسم الوسم `<input />` للتحكم وإجراء التغيير على حقول النموذج من خلال JavaScript .

الخاصية	الشرح
<code>value = " " ;</code>	تستخدم للتحكم بقيمة الخاصية <code>value</code> التي تكتب مع عناصر نماذج الإدخال في لغة <code>html</code>
<code>length ;</code>	تستخدم لتحديد عدد العناصر الموجودة
<code>select () ;</code>	تستخدم لتحديد النص (تظليله)
<code>selectionStart ;</code>	تستخدم مع حقول الإدخال , تعيد رقم يمثل رتبة أول حرف بدأ منه التحديد (التظليل) ويمكن اعطائه رقم ليبدأ منه تظليل النص
<code>selectionEnd ;</code>	تستخدم مع حقول الإدخال , تعيد رقم يمثل رتبة آخر حرف انتها عنده التحديد (التظليل) ويمكن اعطائه رقم لينتهي عنده تظليل النص
<code>blur () ;</code>	تستخدم لإلغاء التركيز من على العنصر
<code>focus () ;</code>	تستخدم للتركيز على عنصر (وضع المؤشر عليه)
<code>size = " " ;</code>	تستخدم للتحكم بحجم العنصر , أي للتحكم بقيمة الخاصية <code>size</code> التي تكتب مع عناصر الإدخال في لغة <code>html</code>
<code>click () ;</code>	لتفعيل حقل أو زر ما وكأنما تم النقر عليه (وقد تلغي تفعيله إذا كان مفعلاً مسبقاً)
<code>checked=" " ;</code>	تستخدم مع خانات الاختيار وتأخذ القيمة <code>true</code> لتحديده والقيمة <code>false</code> لإلغاء تحديده
<code>selectedIndex=" " ;</code>	تستخدم لمعرفة أو تحديد العناصر الموجودة في قوائم الاختيار <code><select></code>
<code>disabled=" " ;</code>	تستخدم هذه الخاصية لتمكين أو إلغاء تمكين المستخدم من الكتابة في الحقل النصي أو الاختيار من عناصر الاختيار وتأخذ القيمة <code>true</code> للتمكين والقيمة <code>false</code> لإلغاء التمكين .
<code>name ;</code>	تستخدم لجلب اسم عنصر من عناصر النموذج
<code>text = " " ;</code>	تستخدم للتحكم (جلب أو تغيير) بالنص الذي يظهر في قائمة الاختيار والذي يكتب بين وسمي البداية والنهاية للعنصر <code><option></code>

كل هذه الخصائص تكتب بنفس الطريقة , لاحظ هذا المثال الذي سنستخدم الخاصية `value` حيث يمكنك تغيير اسم هذه الخاصية بأي واحدة من الخواص الأخرى مع ملاحظة وجود أو عدم وجود القوسين الذين يكتبان أمام الخاصية , مثال /

```
<script LANGUAGE="JavaScript"><!--
```

```
function xx() {
var y = ff.t1.value ;
alert ( y ) ;
ff.t2.value="Hello" ;
}
```

```
//--> </script>
<form name="ff">
<input type="text" name="t1" />
<input type="text" name="t2" />
<input type="button" onclick="xx()" />
</form>
```

ثانياً :- الإطارات

الإطارات <frameset> وهي معروفة في لغة html يمكننا التحكم بها من خلال لغة JavaScript وذلك كما ياي :-

إذا أردنا الإشارة إلى إطار آخر نسبقه بكلمة parent ثم نقطة ثم نذكر اسم الوسم <frame> الذي أعطيناه له من خلال الخاصية name ثم نقطة ثم نكتب اسم الكائن document ثم نقطة ثم نكمل بعده كما تعلمنا مع النماذج , أما إذا أردنا التغيير في نفس الإطار نسبقه بكلمة self ثم نقطة ثم نكتب اسم الكائن document ثم نقطة ثم نكمل بعده كما تعلمنا مع النماذج مع ملاحظة عدم ذكر اسم الإطار هنا لأننا في نفس الإطار .

مثال/ سنكون إطارين في الإطار الأول يوجد حقل نصي وزر وعند النقر على هذا الزر يتم نقل ما كتب في الحقل النصي إلى الحقل النصي الموجود في الإطار الثاني

صفحة الإطار الأول المسماة Ex1.html

```
<html>
<head> </head>
<body>
<form name="f1">
<input type="text" name="t1" />
<input type="button" value="Move"
onclick="parent.frame2.document.f2.t2.value = self.document.f1.t1.value" />
</form>
</body>
</html>
```

صفحة الإطار الثاني المسماة Ex2.html

```
<html>
<head> </head>
<body>
<form name="f2">
<input type="text" name="t2" />
```

```
</form>  
</body>  
</html>
```

```
<html>  
<head> </head>  
<frameset cols="50%,*">  
<frame src="Ex1.html" name="frame1" />  
<frame src="Ex2.html" name="frame2" />  
</frameset>  
</html>
```

الصفحة الرئيسية

الفصل العاشر

(الأحداث EVENTS)

* هذا الجدول يوضح الأحداث المستخدمة

الحدث	الشرح
onClick ()	استجابة لضغط زر يتم تحديده عندها ينفذ جزء من الكود
ondblclick ()	عند النقر بزر الفأرة مرتين متتاليتين
onSubmit ()	عندما يضغط المستخدم على الزر submit
Onmousedown	أثناء ضغط المستخدم على زر معين بالماوس
Onmouseup	عند إطلاق المستخدم للزر الذي ضغطه بالماوس
onMouseOver ()	هو حدث مرور الماوس فوق عنوان او وصلة تشعبية
onMouseOut ()	حدث تحريك لماوس بعيداً عن الوصلة التشعبية
onFocus ()	حدث وضع الماوس على حقل مدخلات معينة
onChange ()	حدث تغيير قيمة معينة لحقل
onBlur ()	حدث ترك حقل البيانات بدون تغيير
onSelect ()	حدث اختيار عنصر من النموذج
onLoad ()	حدث انتهاء متصفح الانترنت من تحميل الصفحة الحالية
onUnLoad ()	حدث الخروج من الصفحة الحالية إلى
onhelp ()	عند ضغط المستخدم على المفتاح f1 من لوحة المفاتيح (طلب مساعدة)
onresize ()	عندما يتم تغيير حجم الصفحة او حجم عنصر ما .
onscroll ()	عند تمرير الإطار الخاص بالصفحة
onkeypress ()	عند الضغط على أي مفتاح من لوحة المفاتيح
onkeydown ()	عند ضغط المستخدم مفتاح من لوحة المفاتيح (أثناء الضغط)
onkeyup ()	عند إطلاق المستخدم للمفتاح الذي ضغطه من لوحة المفاتيح
onabort()	عند قيام المستخدم بالغاء شيء ما .
ondragdrop()	عندما يسحب لمستخدم عنصراً و يلقية (السحب والإفلات) .
onerror()	عندما يكون هناك خطأ برمجي بالجافاسكربت .
onreset()	عند تصفير الخانات .

1- الحدث () onClick

مثال /

```
<html>
<head>
<script LANGUAGE="JavaScript">
function xxx( ) {
alert ( " Hello " ) ; }
</script>
</head>
<body>
<a href="name.html" onClick="xxx ( )" > Hello </a>
</body>
</html>
```

يمكن وضع فارزة منقوطة بعد اسم الدالة (لا يوجد فرق)

2- الحدث () onSubmit

يستخدم هذا الحدث لتنفيذ أمر معين عند الضغط على زر (submit)

مثال /

```
<html>
<head>
<script LANGUAGE="JavaScript">
function xxx( ) {
alert ( " Hello " ) ; }
</script>
</head>
<body>
<form onSubmit="return xxx( )">
<input type="text" />
<input type="submit" />
</form>
</body>
</html>
```

3- الحدثين onMouseOver و onMouseOut

مثال/

```

<html>
<head>
<script LANGUAGE="JavaScript">
function over() {
alert (" Mouse Over " ); }
function out() {
alert (" Mouse Out " ); }
</script>
</head>
<body>
<a href="name.html" onMouseOver="over ()" onMouseOut="out ()" >
Eventmouse </a>
</body>
</html>

```

🔍 ملاحظة // يمكن وضع كود JavaScript مباشرة في الحدث بدون الحاجة إلى كتابته في القسم <head> ومن ثم استدعاء الدالة , لكن في هذه الحالة لا يصح أن نضع وسم البداية والنهاية الخاص بلغة JavaScript داخل الحدث (ويمكن أن نكتب في البداية - داخل الحدث - كلمة JavaScript وبعدها نقطتين ثم نكتب الكود او يمكن أن نكتب الكود مباشرة لا يوجد فرق) , هذه الملاحظة تنطبق على جميع الأحداث .

مثال/

```



```

كما تلاحظ في هذا المثال استخدمنا صورة لكي تظهر عند مرور مؤشر الفأرة فوق الصورة الأولى وذلك من خلال كتابة الاسم الذي أعطيناها لوسم الصورة من خلال الخاصية name والذي هو في المثال mofak وبعده سنضع نقطة ثم اسم خاصية من خصائص الوسم والتي نرغب في أن يجري عليها الحدث وهنا في هذا المثال وضعنا الخاصية src ليتغير مصدر الصورة ويمكن أن نضع width مثلاً او أي خاصية أخرى ويمكن أن نكتب مجموعة من الخواص وذلك بكتابتها في دالة function وبعد ذلك نستدعيها في داخل الحدث , كذلك يمكن أن نغير الصورة عند الضغط على زر معين كما في المثال في الأسفل :-


```

<body>

<form>
<input type="button" onclick="document.mofak.src='name2.jpg' ; " />
</form> </body>

```

هنا استخدمنا في داخل الزر اسم الصورة mofak الذي أعطيناها لها مسبقاً لكن يمكننا أن لا نعطي الصورة اسم وفي داخل الزر سنكتب بدلاً من اسم الصورة mofak القيمة الافتراضية وهي images[0] هنا وضعنا الرقم صفر بين قوسيهما ليشير إلى الصورة الأولى وإذا كانت لدينا صورة ثانية فسيكون تسلسلها 1 والثالثة تسلسلها 2 وهكذا .

-4 الحدث onFocus ()

مثال/

```

<input type="text" onFocus="nameFunction ( ) " />

```

-5 الحدث onChange ()

مثال/

```

<input type="text" onChange="nameFunction ( ) " />

```

-6 الحدث onBlur ()

مثال/

```

<a href="name.html" onBlur="nameFunction ( ) " > Hello </a>

```

-7 الحدثين onLoad () و onUnload ()

مثال/

```

<html>
<head>
<style LANGUAGE="JavaScript" >
function xxx ( ) {
alert ( " Hello " ); }

```

```

</script>
</head>
<body onLoad="xxx ()" onUnLoad="xxx ()" >
</body>
</html>

```

8- onkeyup و onkeydown و onkeypress

مثال/

```

<form name="ff">
<input type="text" name="tt" onkeypress="if (window.event.keyCode== '65' ){
document.ff.tt.value = ' you press A key ' ; } " />
</form>

```

وفي هذا المثال عندما يضغط المستخدم على الحرف A ستظهر له القيمة التالية (you press A key) في الحقل النصي , حيث أننا استخدمنا الرقم الأسكي للمفتاح A ويمكن استخدام رقم أي مفتاح وذلك بكتابة الكود التالي window.event.keyCode ونجعله يساوي رقم المفتاح .

◆ الكائن event

هذا الكائن يستخدم مع الاحداث وتخزن فيه معلومات عن الحدث مثلا احداثيات مؤشر الفأرة ويكتب هذا الكائن بين قوسي الدالة التي سيتم جلبها للحدث (أي أنه سيمرر لها كباراميتير)

```

<html>
<head>
<script LANGUAGE="JavaScript">
function xxx( e ) {
alert ( " e.clientX " );
alert ( " e.clientY " );
}
</script>
</head>
<body>
<p onClick="xxx ()" > Hello </p>
</body>
</html>

```

في هذا المثال سيظهر احدثيات x لمؤشر الفأرة ثم سيظهر احدثيات y للمؤشر , لكن انتبه ان هذا الكلام لا ينفذ مع المتصفح اكسبلورر لان المعلومات التي خزن في الكائن تستخرج بطريقة مختلفة لاحظ :-

```
<html>
<head>
<script LANGUAGE="JavaScript">
function xxx() {
var e = window.event ;
alert (" e.clientX ") ;
alert (" e.clientY ") ;
}
</script>
</head>
<body>
<p onClick="xxx ()" > Hello </p>
</body>
</html>
```

وهذا الكود يعمل فقط مع المتصفح اكسبلورر ولكي نكتب كود ويعمل على كل المتصفحات يجب ان نختبر المتصفح لاحظ هذا المثال :-

```
<html>
<head>
<script LANGUAGE="JavaScript">
function xxx( e ) {
if( window.event ) {
var ev = window.event ;
} else {
var ev = e ;
}
alert (" ev.clientX ") ;
alert (" ev.clientY ") ;
}
</script>
</head>
<body>
<p onClick="xxx ()" > Hello </p>
</body>
</html>
```

ملاحظات عامة

👉 **ملاحظة 1 //** لغة JavaScript حساسة لحالة الأحرف الكبيرة والصغيرة .

👉 **ملاحظة 2 //** في لغة JavaScript ليس من الضروري وضع فارزة منقوطة في نهاية الجملة أي إن وضع الفارزة المنقوطة اختياري , لكن من الأفضل إنهاء الجملة بفارزة منقوطة .

👉 **ملاحظة 3 //** يمكن كتابة التعليق بواسطة شطرتين مائلتين // وذلك إذا كان التعليق سطر واحد , او يمكن وضع التعليق بين (/* التعليق */) إذا كان أكثر من سطر , او يمكن كتابة التعليق مثل التعليق في لغة html هكذا <!-- التعليق -->

👉 **ملاحظة 4 //** كل اكواد JavaScript تتكون من كائنات وأوامر ونفصل بينهم بنقطة .

مثال /

`window.alert (" الرسالة ") ;`

حيث أن window هو الكائن و alert هو الأمر وفصلنا بينهم بنقطة .

👉 **ملاحظة 5 //** لإدراج رموز خاصة مثل (' او " او ; او &) نستخدم الشطرة المائلة للخلف \ .

مثال /

`document.write (" you \& I sing \" happy Eid \" . ")`

* جدول يوضح بعض الأوامر التي تستخدم مع الطباعة

الأمير	وصف الأمر	مثال
\n	سطر جديد	window.alert (" Hi \n Ali ") ;
\t	لترك مسافة وكأنما تم الضغط على المفتاح tab من لوحة المفاتيح	window.alert (" hi \t Ali ") ;
\r	لوضع كل كلمة بسطر ولكن باختلاف position	window.alert (" hi \r Ali ") ;

👉 **ملاحظة 6 //** بعض المتصفحات لا تدعم JavaScript او يكون المستخدم قد عطل ميزة JavaScript في المتصفح فيمكننا أن نستخدم وسم خاص من وسوم HTML لكي يعرض في حال كان المستعرض للصفحة لا يدعم JavaScript أما إذا كان المستعرض يدعم JavaScript فسيعرض كود JavaScript ولن يعرض ما بداخل الوسم وهذا الوسم هو `<noscript>`.

مثال /

```
<html>
<head>
</head>
<body>
<script LANGUAGE="JavaScript"> <!--
document.write ("Last modified : " ) ;
document.write ( document.lastModified ) ;
/--> </script>
<noscript>
your browser can not browse JavaScript !!
</noscript>
</body>
</html>
```

ولاحظ أنه يمكننا أن نضيف ما نشاء من وسوم ودوال وأي شيء نريد عرضه بين الوسامين `<noscript>`

👉 **ملاحظة 7 //** يمكننا عرض كل الخصائص لأي كائن في الصفحة وذلك باستخدام الأمر `prop` والذي هو اختصار لكلمة `properties` وسنكتب هذا الأمر داخل حلقة التكرار `for` `in` ... لعرض جميع الخصائص لهذا الكائن .

مثال / لعرض خصائص الكائن window

```
<script LANGUAGE="JavaScript">
for ( prop in window ) {
document.write ( "window." , prop , " = " , window[prop] , "<br />" ) ; }
</script>
```

ويمكننا كتابة اسم أي كائن من الكائنات بدلاً من الكائن window مثل document أو this وكذلك يمكننا عرض خصائص العناصر الموجودة في الصفحة من form أو button أو text .

مثال/ لعرض خصائص الزر button

```
<body>
<form name="ff">
<input type="button" name="bb" />
</form>
<script LANGUAGE="JavaScript">
for ( prop in ff.bb ) {
document.write ( "button(bb)." , prop , " = " , ff.bb[prop] , "<br />" ) ; }
</script>
</body>
```

👉 **ملاحظة 8 // الكائن json**

يمكننا في جافاسكربت أن ننشأ كائن json بهذا الشكل :-

```
var j = { name1 : "value1" , name2 : "value2" } ;
```

كما تلاحظ هنا نكتب أولاً اسم من أختيارنا ليُمثل اسم الكائن ثم نضع اسم ونسند له قيمة ثم فاروة ثم اسم وقيمته وهكذا , ويمكن أن نستدعي الكائن بمجرد ذكر اسمه وكئنه متغير عادي . كذلك يمكن أن ننشأ الكائن بطريقة أخرى هكذا :-

```
var j = { } ;
j.name1 = "value1" ;
j.name2 = "value2" ;
j.name 3 = "value3" ;
```

ويمكن ان نكتب في القيمة دالة function مجهولة (بدون ذكر اسمها) وستخزن قيمتها في

داخل الكائن

مثال/

```
var j = {
j.name1 = "value1",
j.name2 = function() {
document.writ("value2"); } ,
j.name 3 = "value3"
} ;
```

وكذلك يمكن ان تكون القيمة عبارة عن مصفوفة , مثال /

```
var j = { } ;
j.name1 = "value1" ;
j.name2 = ["A" , "B" , "C"] ;
j.name3 = "value3" ;
```

وأيضاً هنا يمكن أن نستدعي الكائن بمجرد ذكر اسمه وكئنه متغير عادي .
وإذا اردنا التعامل مع قيمة معينة في داخل الكائن , مثلاً اردنا اظهار القيمة الثانية في الكائن
يمكن ان نكتب ذلك بطريقتين :-

```
alert( j.name2 );
```

او يمكن كتابتها بهذا الشكل

```
alert( j['name2'] );
```

او يمكن طباعته بهذا الشكل

```
j.name2( );
```

وإذا اردنا طباعة قيمة معينة من قيمة المصفوفة يمكن كتابتها هذا الشكل

```
alert( j.name2[2]
```

كذلك يمكن تغيير قيمة في داخل الكائن بهذا الشكل

```
j.name2 = "Ahmed" ;
```

ويمكن ايضا اضافة قيمة جديدة الى الكائن هكذا

```
j.Ali = "ABC" ;
```

ملاحظة 9 // اذا كان لدينا في كود الـ HTML الوسم <iframe> وأردنا تحديد محتوياته (أي
الوسوم في داخله كإضافة خاصية او وسم في داخله) فأولاً يجب أن نعرف ان هذا الوسم يحمل
صفحة تعتبر خارج صفحتنا فمثلاً لتحديد الوسم <body> الموجود داخله لاحظ هذا المثال

```
var dir_body = document.getElementById('id_iframe').contentDocument.body ;
```