بسم الله الرحمن الرحيم

سبحانك لا علم لنا إلا ما علمتنا إنك أنت العليم الحكيم

المقدمة :

الحمد لله ماحمده الحامدون وغفل عن حمده الغافلون والصلاة والسلام علي عبده ورسوله محمد صلاة بعدد ذرات الخلائق وما يكون . ورضاك اللهم عن آله الطيبين وصحبه المكرمين المبجلين أجمعين وبعد ،

يقدم هذا الجزء من الكتاب طريقة إستخدام لغة الجافا سكربت في برمجة الويب

في حالة وجود أي أخطاء أرجو اعلامي عن الخطأ علي العنوان التالي
a_elhussein@hotmail.com

وأرجو من كل من أستفاد من هذا الكتاب أن يدعوا لي بالتوفيق في الدنيا والأخرة

إهداء :

أهدي هذا الكتاب إلي الجيل القادم الذي يعز الله به الإسلام

وإنا مادامت فيا الحياة باذل جهدي وعقلي ومستفرغ طاقتي في العلم وذلك لثلاثة أمور

- إفادة من يطلب العلم في حياتي وبعد مماتي
- ذخيرة لي في قبري ويوم حسابي
- رفعة لسلطان المسلمين

تأليف : الحسين محمد علي

APPROVED

المحتويات

الفصل الأول : المصفوفات Arrays ٣
الفصل الثاني : المصفوفات ذات الأبعاد المتعددة Multidimensional Arrays ١٧
الفصل الثالث : التعامل مع النصوص ٣٧
الفصل الرابع : كائن النصوص ٤٧
الفصل الخامس : التعامل مع التعبيرات المنتظمة Regular Expressions ٦٣
الفصل السادس : التعامل مع التاريخ ٧٨

الفصل الأول

المصفوفات

Arrays

سوف نناقش إن شاء الله في هذا الفصل النقاط التالية :

- الكائنات في الجافا سكربت
- ما هي المصفوفات
- إنشاء مصفوفة
- المصفوفات بشكل عام
- تمثيل المصفوفة
- ملء المصفوفة بالقيم
- المصفوفات المجمعة Associative Arrays
- حذف عنصر من مصفوفة

الكائنات في الجافا سكربت

تطور أسلوب البرمجيات علي مدي ٥٠ عام وقد أثمر هذا التطور علي ظهور طرق واستراتيجيات مختلفة بهدف تكوين برمجيات عالية الجودة ولتقليل الوقت والجهد المستهلك في تطويرها إلى اقل حد ممكن .
وطريقة تطوير البرمجيات الأكثر نجاحا و شيوعا في الاستخدام اليوم هي الطريقة **الموجهة للكائنات** Object Oriented Programming .

فهذه الطريقة تشكل عناصر البرمجة أو التطبيق على صوره **كائن** Object تعرف **خواصه وطرقه** وبعد ذلك يمكن استخدامه لأداء مهام خاصة كما سنري إن شاء الله بالجزء الثالث من هذا الكتاب .

لكننا للأسف لا نستطيع القول بأن لغة الجافا سكربت تدعم برمجة الكائنات الموجهة كما هو الحال بلغة السي بلس بلس أو الجافا كما سوف يتضح بالجزء الثالث من هذا الكتاب ، ولكننا يمكننا القول بأن لغة الجافا سكربت تدعم **برمجة الكائنات الأساسية او الإعتمادية** Object Base أي إعتمادنا علي إستخدام كائنات مبنية داخل لغة الجافا سكربت built-in Objects مثل الكائن window و الكائن document .

كما نعلم أن لغة الجافا سكربت تم تصميمها لبرمجة صفحات الويب ، لذلك تم بناء الكائنات الأساسية للغة الجافا سكربت حتي توافق مميزات متصفحات الأنترنت (مثل متصفح أنترنت أكسبلورار) مثل الكائن window و document ، بالإضافة إلي أن لغة الجافا سكربت توفر بعض الكائنات الأخري المفيدة مثل الكائن Date و Array و String .

ولكن ما هو الكائن Object

بشكل مختصر:
الكائن هو متغير مركب ينشأ من قالب class هذا القالب يحدد الخواص والطرق المميزة للكائنات الناشئة منه .

ويتكون الكائن من
- خصائص
- طرق أو وظائف
- أحداث

علي سبيل المثال نفترض أن هناك كائن يمثل سيارة "عربة" هذه السيارة لها التالي :
- **خصائص** Properties
  مثل لونها ، موديلها
- **طرق أو وظائف** Methods
  ولها عدة وظائف تؤديها مثل أنها تمشي وتقف و تستدير

ما هي المصفوفات

قبل أن نذهب لتعريف ما هي المصفوفات ، هب أننا نريد عمل التالي :
نريد طباعة الرسائل التالية
"**مرحبا بك**"
"**نحن الأن نتعلم المصفوفات**"
"**الحمد لله**"

فعلي حسب ما تعلمناه في فصل المتغيرات
فسوف نقوم بتعريف ثلاث متغيرات حتي نحفظ فيها الرسائل السابقة ، كما يلي

```
var Msg1 = "مرحبا بك";
var Msg2 = "نحن الأن نتعلم المصفوفات";
var Msg3 = "الحمد لله";
```

ثم نقوم بطباعة هذه الرسائل كما يلي

```
alert( Msg1 );
alert( Msg2 );
alert( Msg3 );
```

تخيل أنك تريد عمل المثال السابق ولكن ليس علي ثلاث متغيرات بل علي ١٠٠ متغير نصي أو قل عدد غير محدد من المتغيرات ،ربما يسبب لك هذا إحساس بالضيق لكثرة الأكواد التي سوف تكتب لإتمام هذه المهمة

من هنا أتت الحاجة لعمل نوع جديد من المتغيرات وهو ما يطلق عليه **المصفوفات** Arrays

ولكن ما هي المصفوفات Arrays

المصفوفات هي من إحدي أنواع المتغيرات ولكن يمكنك أن تخزن بهذا المتغير قيمة واحدة أو أكثر .

إنشاء مصفوفة

يمكننا إنشاء المصفوفة بعدة طرق كما يلي

١- **إنشاء مصفوفة فارغة** (**لا تحتوي علي عناصر**)

```
var myArray = new Array();
```

٢- **إنشاء مصفوفة مكونة من عناصر عددها** n (حيث n تمثل عدد صحيح موجب)

```
var myArray = new Array(n);
```

إي يمكنك تعريف مصفوفة مكونة من خمس عناصر كما يلي

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

٣- **إنشاء مصفوفة وملء عناصرها في نفس الوقت**

لإنشاء مصفوفة وملء عناصرها في نفس الوقت يوجد عدة طرق لإتمام ذلك :

أ -

```
var myArray = new Array("item1"," item2"," item3");
```

إي يمكنك تعريف مصفوفة وإعطاء قيم لعناصرها بشكل مبدئي كما بالمثال التالي
سوف نقوم بإنشاء مصفوفة تسمي empArray تحتوي علي اسماء ثلاث موظفين

```
var empArray = new Array("محمد عبد الله","حمدي غانم","أحمد محسن");
```

ب -

```
var myArray = new Array["item1"," item2"," item3"];
```

أو بدون إستخدام كلمة new Array كما يلي

```
var myArray = ["item1"," item2"," item3"];
```

إي يمكنك تعريف مصفوفة وإعطاء قيم لعناصرها بشكل مبدئي كما بالمثال التالي
سوف نقوم بإنشاء مصفوفة تسمي empArray تحتوي علي اسماء ثلاث موظفين

```
var empArray = new Array["محمد عبد الله","حمدي غانم","أحمد محسن"];
```

أو

```
var empArray = ["محمد عبد الله","حمدي غانم","أحمد محسن"];
```

المصفوفات بشكل عام

المصفوفة نستطيع تشبيهها بعمارة . العمارة يحتوي كل طابق منها على شقة واحدة .
لنفرض أن العمارة تتكون من أربعة طوابق
فكان الطابق الأول يسكن به الحسين والطابق الثاني يسكن به إسماعيل والطابق الثالث يسكن به إبراهيم والطابق الرابع يسكن به يوسف .

إذا هنا لدينا عمارة تتكون من أربعة طوابق كل طابق يحتوي على شخص فهذا هو الحال بالنسبة للمصفوفة فالعمارة هي **أسم المصفوفة** . وعدد الطوابق الأربعة هو **عدد عناصر المصفوفة** والتى هي أربعة والاشخاص الذي كان كل شخص منهم يسكن بطابق هم **قيمة كل عنصر في المصفوفة** .

دعنا نطبق هذا المثال بشكل برمجي

كلمة عمارة لنختصرها ونسميها arr

```
var arr = new Array("الحسين","إسماعيل","إبراهيم","يوسف");
```

إذا هنا الموقع الأول في المصفوفة يساوي الحسين والموقع الثاني يساوي إسماعيل والموقع الثالث يساوي إبراهيم والموقع الرابع يساوي يوسف .

ولكن عادة في المصفوفات **نبدأ من الصفر وليس من الواحد** أي نقول موقع الصفر يساوي الحسين و الموقع الاول يساوي إسماعيل والموقع الثاني يساوي إبراهيم والموقع الثالث يساوي يوسف .

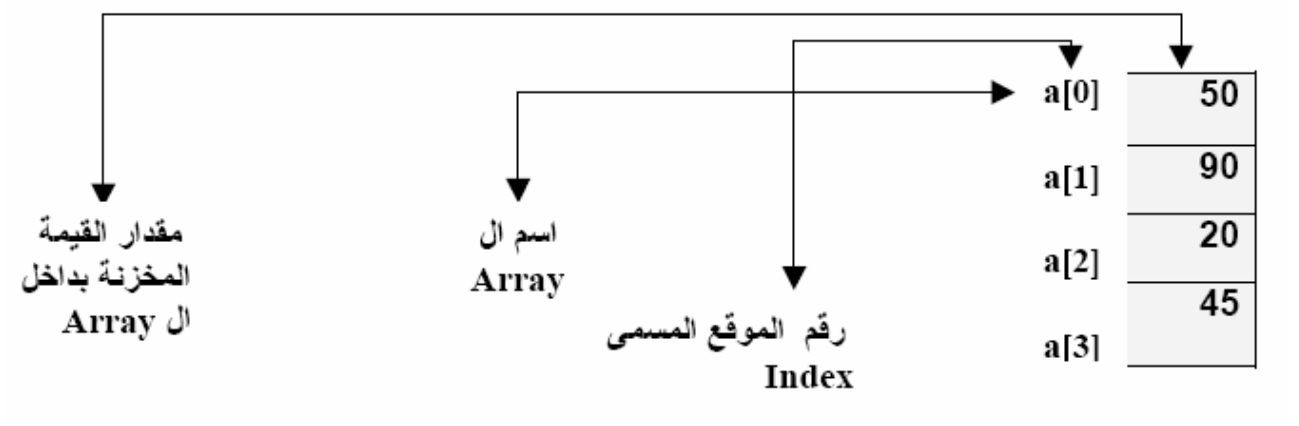
لعلك إنزعجت من ذلك ، لا داعي لهذا الإنزعاج أعتبر العمارة تتكون من طابق أرضي وطابق أول وطابق ثاني وطابق ثالث ، إذا الحسين سوف يسكن في الطابق الارضي وهو الصفر أي موقع رقم صفر في المصفوفة ، وإسماعيل في الطابق الأول أي الموقع الاول في المصفوفة وهكذا .

تمثيل المصفوفة

نقصد بتمثيل المصفوفة اي كيفية تمثيل المصفوفة بداخل ذاكرة الجهاز
فعلي سبيل المثال كيف يتم تمثيل المصفوفة التالية بذاكرة الجهاز

```
var a = new Array(50,90,20,45);
```

يوضح الشكل التالي كيفية تمثيل المصفوفة السابقة



ملء المصفوفة بالقيم

كما تعلمنا سابقا كيفية إنشاء مصفوفة وملء عناصرها في نفس الوقت ، يمكننا أيضا إنشاء مصفوفة ثم ملء عناصرها بعد ذلك كما يلي

```
var arr = new Array(3);

// نقوم بملء المصفوفة
arr[0] = 100;
arr[1] = 30;
arr[2] = 230;
```

كما بالمثال السابق تم تحديد مصفوفة مكونة من ثلاث عناصر كما يلي

```
var arr = new Array(3);
```

ثم قمنا بتحديد القيم المخزنة بكل عنصر من عناصر المصفوفة إبتداء من العنصر الموجود **بالموقع صفر** كما يلي

```
// نقوم بملء المصفوفة
arr[0] = 100;
arr[1] = 30;
arr[2] = 230;
```

ولتقليل حجم البرنامج يمكننا إستخدام حلقات التكرار لملء المصفوفات كما بالمثال التالي

```
var arr = new Array(100);

// نقوم بملء المصفوفة
for( var i = 0; i<3; i++ ){
     arr[i] = 30;
}

for( i = 3; i<6; i++ ){
     arr[i] = 13;
}

for( i = 6; i<100; i++ ){
     arr[i] = 40;
}
```

وبهذه الطريقة سوف يتم تحديد ١٠٠ عنصر للمصفوفة arr كما يلي

```
var arr = new Array(100);
```

ثم يتم تخزين القيمة ٣٠ بداخل الثلاث عناصر الأولي للمصفوفة كما يلي

```
for( var i = 0; i<3; i++ ){
     arr[i] = 30;
}
```

ثم يتم تخزين القيمة ١٣ بالعناصر الثلاث التالية كما يلي

```
for( i = 3; i<6; i++ ){
     arr[i] = 13;
}
```

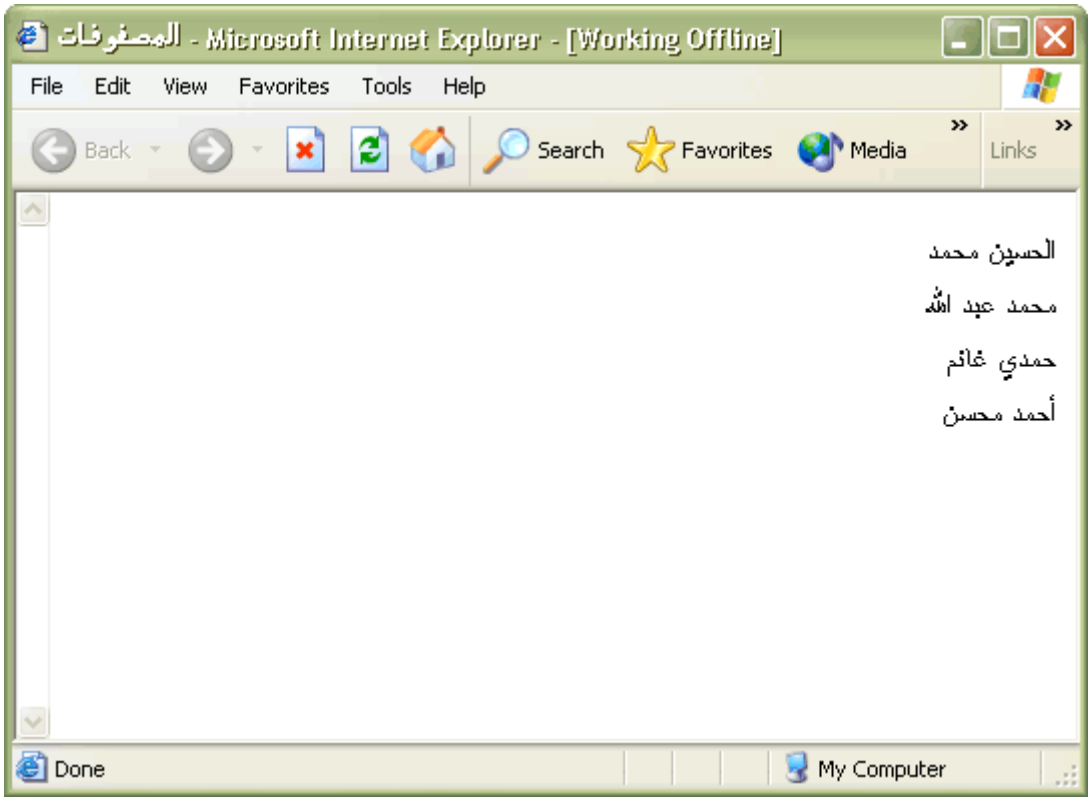
ثم يتم تخزين القيمة ٤٠ بالعناصر التالية للمصفوفة كما يلي

```
for( i = 6; i<100; i++ ){
     arr[i] = 40;
}
```

تمرين كتابة محتويات المصفوفة

```
<HTML dir=rtl>
  <Title> المصفوفات </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
            var empNameArray = new Array("الحسين محمد","محمد عبد الله","حمدي غانم","أحمد محسن");
            // var empNameArray = ["الحسين محمد","محمد عبد الله","حمدي غانم","أحمد محسن"];

            for( var i=0; i<4; i++){
               document.write(empNameArray[i] );
               document.write( "<br> " );
            }
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



إستخدام جملة for in التكرارية

كما قد كنا أشرنا بالجزء الأول من هذا الكتاب أن جملة for in تستخدم لعمل تكرار للمتغيرات من النوع الكائني objects مثل المصفوفات Arrays

لاحظ بالمثال السابق عندما قمنا بإستخدام جملة for ، فقد إحتجنا لتحديد عدد مرات التكرر بمقدار يساوي طول المصفوفة (اي بعدد عناصر المصفوفة) كما يلي

```
var empNameArray = new Array("أحمد محسن","حمدي غانم","محمد عبد الله","الحسين محمد");

for( var i=0; i<4; i++){
     document.write(empNameArray[i] );
     document.write( "<br> " );
}
```

ولكننا عند إستخدامنا لجملة for in لا نحتاج لتحديد عدد مرات التكرار ، لأنها بشكل تلقائي سوف يتم التكرار بعدد عناصر المصفوفة .

الصيغة العامة

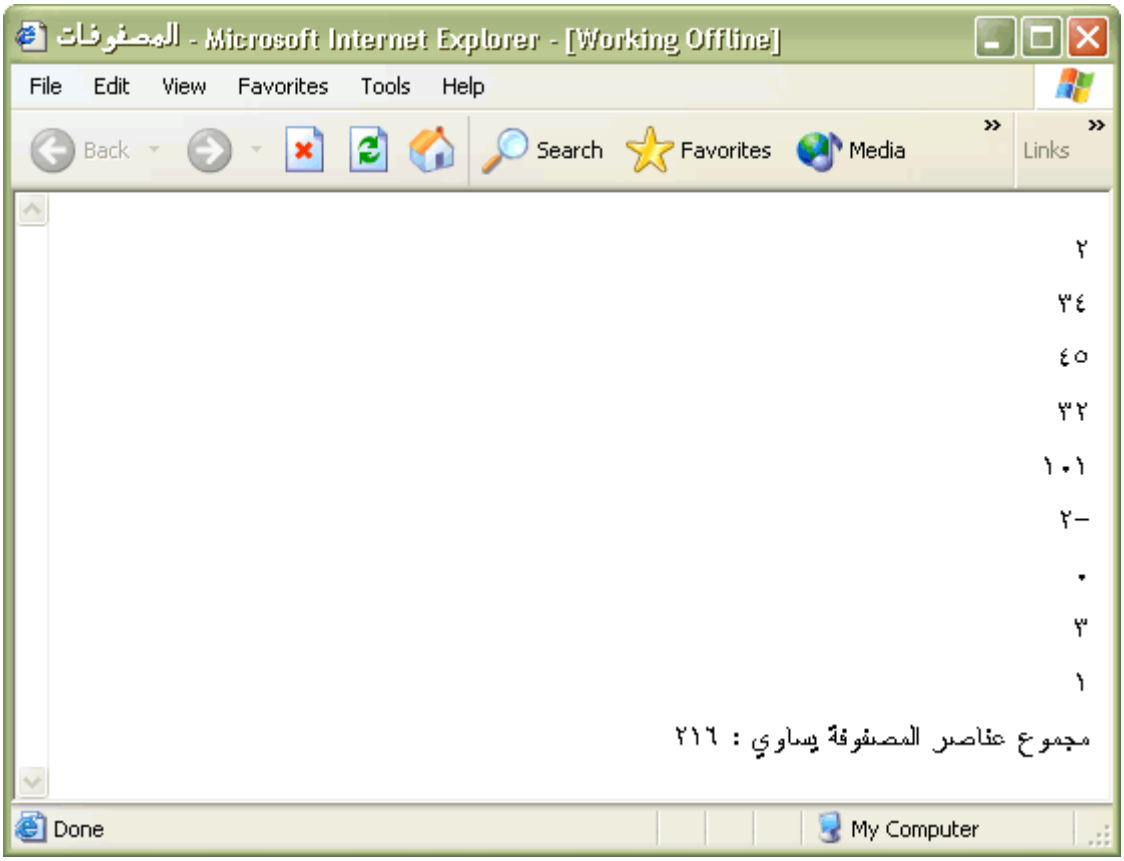
```
for ( المتغير in الكائن) {
        // الأكواد
}
```

مثال توضيحي

لحساب مجموع قيم المصفوفة

```
<HTML dir=rtl>
  <Title> المصفوفات </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
           var numArray = new Array(2,34,45,32,101,-2,0,3,1);
           var sum = 0;

           for( var index in numArray ){
              sum += numArray[index];

              document.write(numArray[index] );
              document.write( "<br> " );
           }

           document.write(" مجموع عناصر المصفوفة يساوي : " + sum);
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



مثال

نريد عمل دالة يمرر لها اي عدد من المعاملات ثم تقوم بإرجاع القيمة الأكبر لهذه القيم
وقد تم الإشارة لهذا المثال سابقا بالجزء الأول من الكتاب كما يلي

```
<HTML dir=rtl>
  <Title> تمرير القيم إلي الدوال </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    // تعريف الدالة getMax
    function getMax () {
         var args = getMax.arguments;
         var max = args[0];

         for( var i=1; i<args.length; i++ ){
```

```
            if( max < args[i] )
                max = args[i];
        }

        return max;
    }

     alert( getMax(21,34) );
     alert( getMax(2,43,5) );
     alert( getMax(2,1,3,4,7,4,8,1) );

    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ولذلك قمنا بتعريف الدالة getMax

```
    function getMax () {
        var args = getMax.arguments;
        var max = args[0];

        for( var i=1; i<args.length; i++ ){
            if( max < args[i] )
                max = args[i];
        }

        return max;
    }
```

وهذه الدالة يمكنها أستقبال اي عدد من المعاملات عند إستدعائها كما يلي

```
     alert( getMax(21,34) );
     alert( getMax(2,43,5) );
     alert( getMax(2,1,3,4,7,4,8,1) );
```

ويتم إستقبال المعاملات الممررة لهذة الدالة في المصفوفة التابعة لكائن هذة الدالة
وهذه المصفوفة تسمي arguments وهي أحدي خصائص الدالة getMax كما يلي

```
    function getMax () {
        var args = getMax.arguments;

        .
        .
        .
    }
```

وبذلك يكون المتغير args يشير إلي مصفوفة عناصرها هي المعاملات الممررة للدالة .

المصفوفات المجمعة Associative Arrays

كما تعلمنا سابقا التعامل مع عناصر المصفوفة يتم عن طريق تحديد موقع هذا العنصر فعلي سبيل المثال لتعين قيمة العنصر الأول بالمصفوفة arr نقوم بإستخدام موقع هذا العنصر بالمصفوفة وهو **صفر** لذلك لتعين قيمته نكتب arr[0] كما يلي

```
var arr = new Array("أحمد محسن","حمدي غانم","محمد عبد الله","الحسين محمد");

// العنصر الأول بها
document.write( arr[0] );
```

وبذلك نخلص إلي أنه يتم التعامل مع عناصر المصفوفة من خلال مواقعها بالنسبة للمصفوفة

أما بالنسبة **للمصفوفات المجمعة** فلا يتم التعامل مع عناصرها من خلال مواقع تلك العناصر ، بل يتم التعامل من خلال أستخدام **كلمات مفتاحيه** keys حتي يمكننا الإشارة لعنصر ما بالمصفوفة .

طريقة ملء المصفوفات المجمعة

```
var earth = new Array();

earth["diameter"] = "7920 miles";
earth["distance"] = "93 million mile";
earth["year"] = "365.25 days"
earth["day"] = "24 hours";
```

يمكننا أيضا التعامل مع الكلمات المفتاحية كأنها خصائص تم إضافتها إلي كائن المصفوفة (كما سوف يتضح بشكل مفصل بالجزء الثالث من هذا الكتاب) كما يلي

```
var earth = new Array();

earth.diameter = "7920 miles";
earth.distance = "93 million mile";
earth.year = "365.25 days"
earth.day = "24 hours";
```

لاحظ أننا لا نستطيع الإشارة لعناصر المصفوفة بواسطة موقع العنصر بالنسبة للمصفوفة كما يلي

```
var earth = new Array();

earth.diameter = "7920 miles";

alert( earth[0] );
```

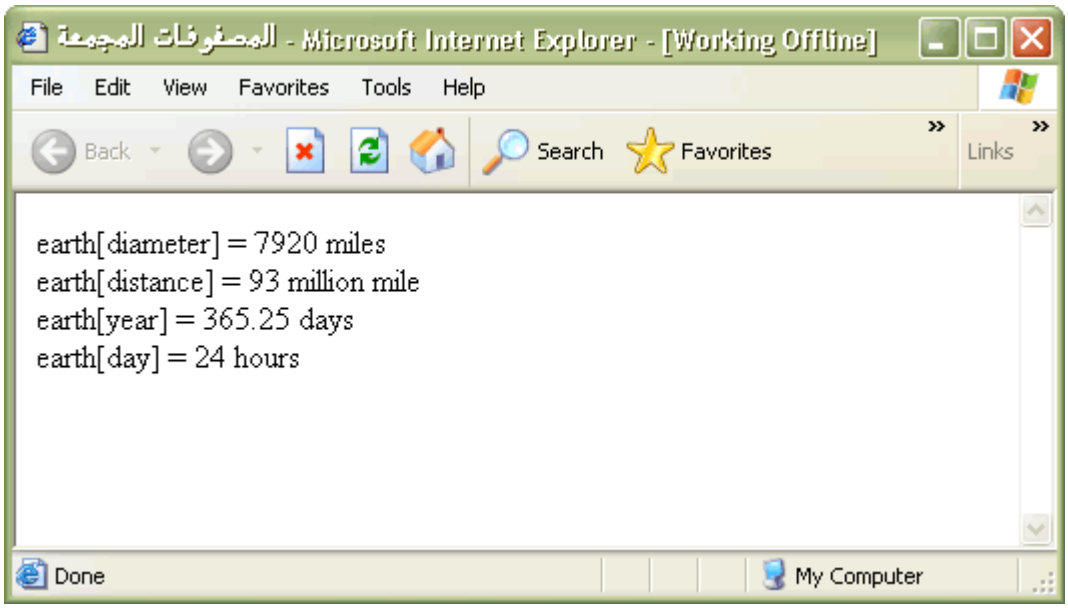
يمكننا إستخدام جملة for in لتعين قيم عناصر المصفوفة المجمعة كما يلي

مثال

نريد كتابة محتويات مصفوفة مجمعة بالمتصفح

```
<HTML>
  <Title> المصفوفات المجمعة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    // مصفوفة مجمعة تعريف
    var earth = new Array();

    earth["diameter"] = "7920 miles";
    earth["distance"] = "93 million mile";
    earth["year"] = "365.25 days"
    earth["day"] = "24 hours";

    for( var key in earth ){
            document.write ( "earth[" + key + "]" );
            document.write ( " = " );
            document.write ( earth[key] + "<br>" );
    }
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



حذف عنصر من مصفوفة

يمكننا حذف عنصر من عناصر المصفوفة بإستخدام **المعامل** delete
وهو أحد المعاملات الخاصة للغة الجافا سكربت .

لذلك إذا أردنا حذف العنصر الثالث بالمصفوفة oceans نقوم بعمل التالي

```
var oceans = new Array("Atlantic","Pacific"," Indian"," Arctic");

delete oceans[2];  //  لحذف العنصر الثالث بالمصفوفة
```

ما هي التغيرات التي حدثت بالمصفوفة
- أولا تم حذف العنصر الثالث بالمصفوفة .

فإذا قمنا بتعين قيمة العنصر الثالث للمصفوفة سوف يعطينا القيمة undefined كما يلي

```
var oceans = new Array("Atlantic","Pacific"," Indian"," Arctic");

delete oceans[2];  //  لحذف العنصر الثالث بالمصفوفة

alert( " oceans[2] = " + oceans[2] );
```

- لكن لاحظ أن **طول المصفوفة لم يتغير** فمازال طول المصفوفة يساوي ٤
وتكون المصفوفة كما بالشكل التالي

```
oceans[0] = "Atlantic";
oceans[1] = "Pacific";
oceans[3] = "Arctic";
```

الفصل الثاني

المصفوفات ذات الأبعاد المتعددة

Multidimensional Arrays

سوف نناقش إن شاء الله في هذا الفصل النقاط التالية :

- المصفوفات ذات الأبعاد المتعددة Multidimensional Arrays
- مصفوفة المصفوفة Array of Array
- كائن المصفوفة Array Object
  - خصائص كائن المصفوفة Array Object Properties
  - دوال كائن المصفوفة Array Object Methods

المصفوفات ذات الأبعاد المتعددة Multidimensional Arrays

دعنا نقول أنه يوجد نوعين من المصفوفات أحدهما :

- **ذات بعد واحد** وهو ما كنا أشرنا إليه سابقا كما بالمثال التالي

```
oceans[0] = "Atlantic";
oceans[1] = "Pacific";
oceans[2] = "Indian";
oceans[3] = "Arctic";
```

فالمصفوفة oceans تمثل مصفوفة ذات بعد واحد
أي أنه يمكن تمثيل عناصر المصفوفة علي شكل خطي كما بالشكل التالي

Atlantic	**Pacific**	**Indian**	**Arctic**

- والأخري **ذات أبعاد متعددة** وهو ما سوف نتحدث عنه

أولا المصفوفة ذات الأبعاد المتعددة مثالا علي ذلك المصفوفة ثنائية الأبعاد ، أي أنه يمكن تمثيل عناصر المصفوفة علي شكل مستطيلي ، وأيضا المصفوفات ثلاثية الأبعاد يمكن تمثيلها في شكل مكعبي
ولن أطيل في هذا الحديث لأنه **للأسف لا توفر لنا لغة الجافا سكربت امكانية عمل مصفوفة متعددة الأبعاد**
ولكن أنتظر قليلا فإنه يمكن التحايل علي هذا الموضوع بعمل ما يسمي **مصفوفة المصفوفة**

مصفوفة المصفوفة Array of Array

مصفوفة المصفوفة أو ما تسمي أحيانا بمصفوفة الجاجد Jagged Array وهي طريقة للتحايل لعمل مصفوفة متعددة الأبعاد .

وتكون فكرة عمل تلك المصفوفة معتمد علي أننا نقوم بعمل التالي :

- نقوم بإنشاء مصفوفة ذات بعد واحد كما تعلمنا سابقا
- ثم نقوم بملء عناصر تلك المصفوفة ، ولكن أنتظر قليلا فأننا لن نقوم بملء عناصر المصفوفة بقيم من نوع نصي أو رقمي كما كان بالسابق ، ولكننا سوف نجعل كل عنصر من عناصر تلك المصفوفة يشير إلي مصفوفة أخري ، وربما تكون المصفوفة المشارة إليها ذات بعد واحد أو مصفوفة من نوع مصفوفة المصفوفة

دعنا نأخذ مثال عملي علي ذلك
هب أننا نريد تمثيل مصفوفة لشركة لها عدة فروع وكل فرع له الخصائص التالية "الاسم" و "العنوان" و "نشاطه"
كما نري هنا أننا نملك مصفوفة من الفروع (لأن الشركة ربما يكون لها أكثر من فرع) ، إذا نقوم بعمل مصفوفة ذات بعد واحد ونطلق عليها مثلا CompBranches ، ولكن يأتي بعد ذلك كيفية ملء عناصر تلك المصفوفة ، فكما تم الأشارة سابقا أن كل فرع (وهو ما يمثل عنصر بمصفوفة الفروع) تمثل قيمته من خلال اسم الفرع وعنوانه ونشاطه
إذا نحتاج لعمل مصفوفة جديد تحمل بيانات الفرع ويكون طولها ٣ كما يلي

```
var CompBranches = new Array(2);

CompBranches[0] = new Array("الفرع الرئيسي","القاهرة" , "التجارة");
CompBranches[1] = new Array("فرع المنيرة","الأسكندرية" , "المراجعة");
```

كما نري فقد قمنا بتعريف المصفوفة CompBranches التي تعبر عن فروع الشركة كما يلي

```
var CompBranches = new Array(2);
```

ثم قمنا بملء بيانات كل فرع علي حدي كما يلي

```
CompBranches[0] = new Array("الفرع الرئيسي","القاهرة" , "التجارة");

CompBranches[1] = new Array("فرع المنيرة","الأسكندرية" , "المراجعة");
```

كما يمكننا عمل المثال السابق بعدة طرق كما يلي

```
var CompBranches = new Array(2);

var branch1 = new Array("الفرع الرئيسي","القاهرة" , "التجارة");
CompBranches[0] = branch1;

var branch2 = new Array("فرع المنيرة","الأسكندرية" , "المراجعة");
CompBranches[1] = branch2;
```

أو كما يلي

```
var CompBranches = [ ["الفرع الرئيسي","القاهرة" , "التجارة"] , ["فرع المنيرة","الأسكندرية" , "المراجعة"] ]
```

أو كما يلي

```
var CompBranches = new Array(2);

var branch1 = new Array(3);
branch1[0] = "الفرع الرئيسي";
branch1[1] = "القاهرة";
branch1[2] = "التجارة";
CompBranches[0] = branch1;

var branch2 = new Array(3);
```

```
    branch2[0] = "فرع المنيرة";
    branch2[1] = "الأسكندرية";
    branch2[2] = "المراجعة";
    CompBranches[1] = branch2;
```

أو كما يلي

```
    var CompBranches = new Array(2);

    CompBranches[0] = new Array(3);
    CompBranches[0][0] = "الفرع الرئيسي";
    CompBranches[0][1] = "القاهرة";
    CompBranches[0][2] = "التجارة";

    CompBranches[1] = new Array(3);
    CompBranches[1][0] = "فرع المنيرة";
    CompBranches[1][1] = "الأسكندرية";
    CompBranches[1][2] = "المراجعة";
```

يمكننا إستخدام المصفوفة المجمعة Associative Array كما يلي

```
    var CompBranches = new Array(2);

    CompBranches[0] = new Array(3);
    CompBranches[0]["name"] = "الفرع الرئيسي";
    CompBranches[0]["address"] = "القاهرة";
    CompBranches[0]["activity"] = "التجارة";

    CompBranches[1] = new Array(3);
    CompBranches[1]["name"] = "فرع المنيرة";
    CompBranches[1]["address"] = "الأسكندرية";
    CompBranches[1]["activity"] = "المراجعة";
```

مثال لطباعة بيانات مصفوفة الفروع السابقة بإستخدام جملة for

```
<HTML>
  <Title> مصفوفة المصفوفة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var CompBranches = new Array(2);

    CompBranches[0] = new Array(3);
    CompBranches[0][0] = "الفرع الرئيسي";
    CompBranches[0][1] = "القاهرة";
    CompBranches[0][2] = "التجارة";

    CompBranches[1] = new Array(3);
```

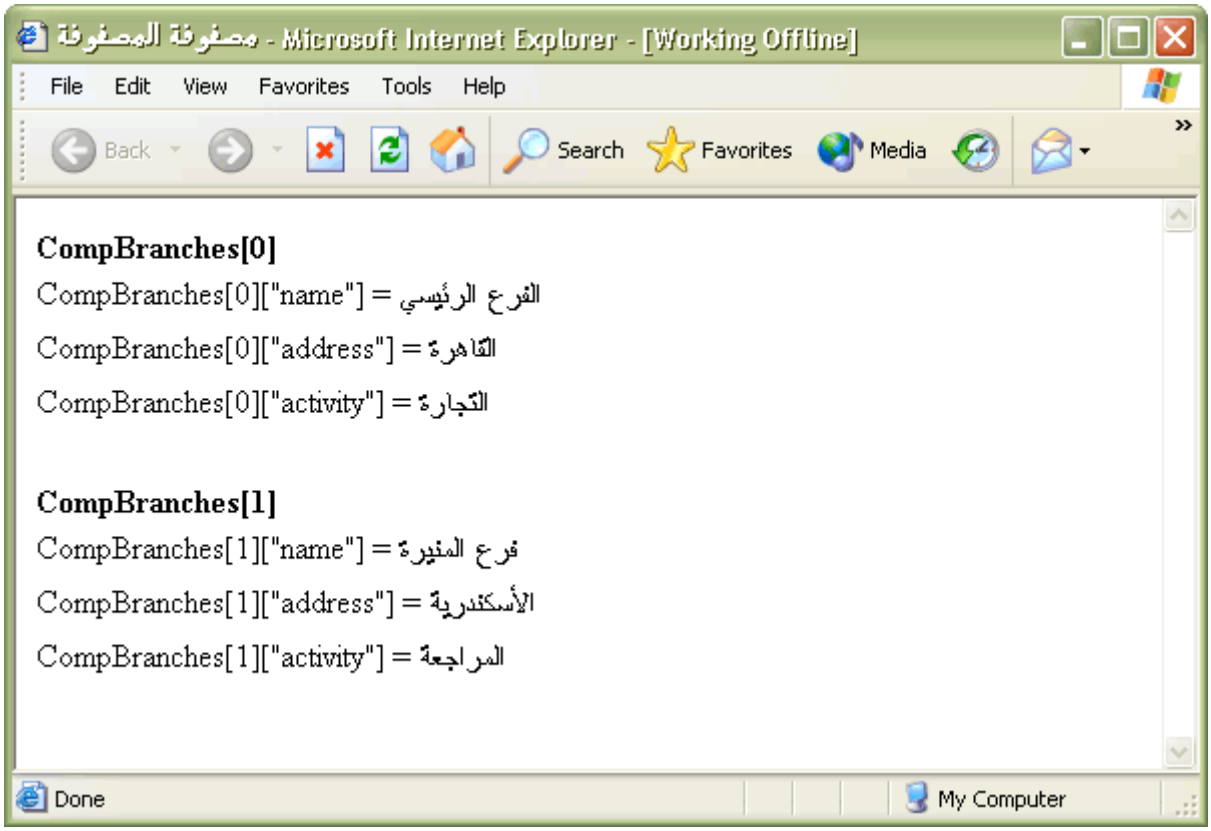
```
    CompBranches[1][0] = "فرع المنيرة";
    CompBranches[1][1] = "الأسكندرية";
    CompBranches[1][2] = "المراجعة";

    for( var i=0; i<2; i++){
         document.write( "<b>CompBranches[" + i + "]</b>" + "<br>" );

         for(var j=0; j<3; j++){
             document.write( " CompBranches[" + i + "][" + j + "] = " );
             document.write( CompBranches[i][j] + "<br>" );
         }

         document.write( "<br>" );
    }
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :

مثال أخر لطباعة بيانات مصفوفة الفروع السابقة بإستخدام جملة for in

```
<HTML>
  <Title> مصفوفة المصفوفة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var CompBranches = new Array(2);

    CompBranches[0] = new Array(3);
    CompBranches[0]["name"]     = "الفرع الرئيسي";
    CompBranches[0]["address"]  = "القاهرة";
    CompBranches[0]["activity"] = "التجارة";

    CompBranches[1] = new Array(3);
    CompBranches[1]["name"]     = "فرع المنيرة";
    CompBranches[1]["address"]  = "الأسكندرية";
    CompBranches[1]["activity"] = "المراجعة";

    for( var i=0; i<2; i++){
         document.write( "<b>CompBranches[" + i + "]</b>" + "<br>" );

         for(var key in CompBranches[i] ){
             document.write( " CompBranches[" + i + "][\"" + key + "\"] = " );
             document.write( CompBranches[i][key] + "<br>" );
         }

         document.write( "<br>" );
    }
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



كائن المصفوفة Array Object

دعنا نتحدث الأن عن الخصائص و الدوال الخاصة بكائن المصفوفة

خصائص كائن المصفوفة Array Object Properties

الخاصية constructor

سوف نتحدث عن الخاصية constructor بالجزء الثالث من هذا الكتاب ، عند الحديث عن الكائنات والبرمجة الكائنية

الخاصية prototype

سوف نتحدث عن الخاصية prototype بالجزء الثالث من هذا الكتاب ، عند الحديث عن الكائنات والبرمجة الكائنية والوراثة Inheritance

الخاصية length

تستخدم الخاصية length لحساب طول المصفوفة اي عدد عناصر المصفوفة

مثال

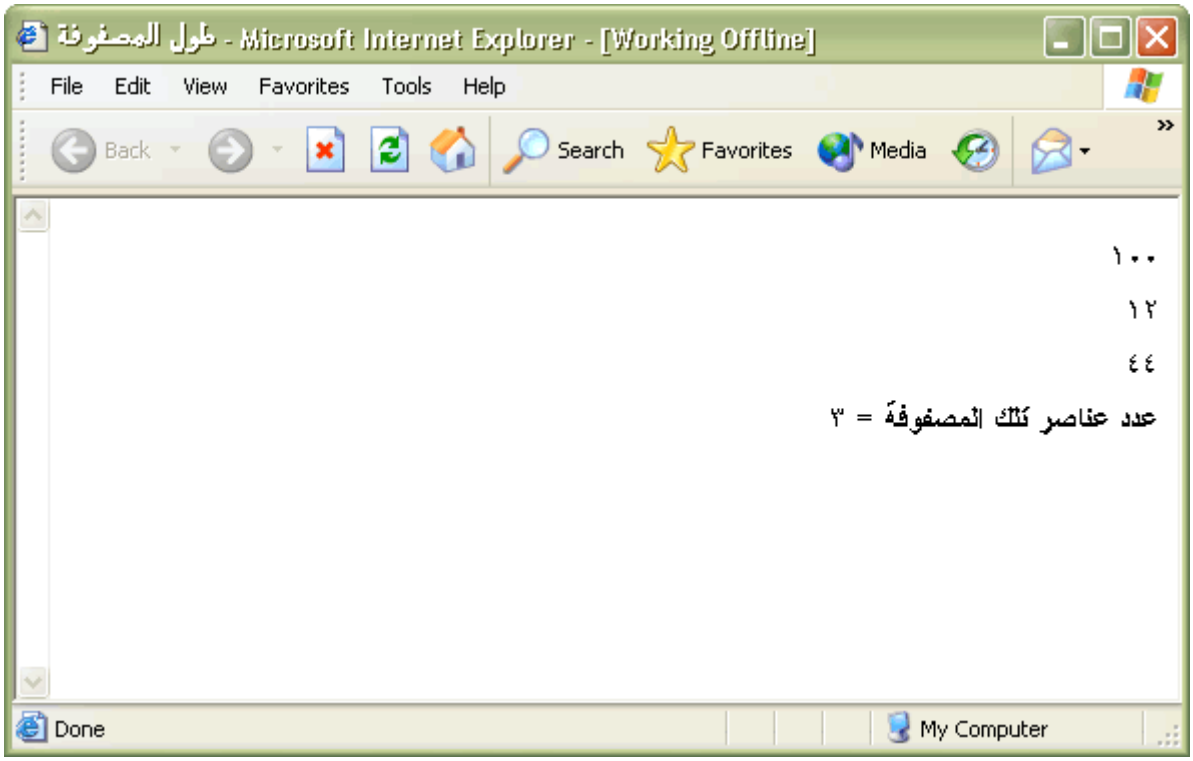
```
var arr = new Array("100","12","44");

alert( arr.length );
```

ويكون الناتج كما يلي

مثال

```
<HTML dir=rtl>
  <Title> طول المصفوفة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var arr = new Array();

    arr[0] = "100";
    arr[1] = "12";
    arr[2] = "44";

    for( var i=0; i<arr.length; i++){
         document.write( arr[i]);
         document.write( "<br>" );
    }

    document.write( "<b>  = عدد عناصر تلك المصفوفة </b>" + arr.length );
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



مثال

```
var arr = new Array("100","12","44");

alert( arr.length );  // طول المصفوفة يساوي ٣

// إضافة عنصر جديد للمصفوفة
arr[ arr.length ] = "200";

alert( arr.length );  // طول المصفوفة يساوي ٤
```

ويكون الناتج كما يلي

دوال كائن المصفوفة Array Object Methods

الدالة concat

تستخدم لدمج مصفوفتين في مصفوفة جديدة

الصيغة العامة

```
Array.concat( arrayObject );
```

مثال

```
var array1 = new Array(1,2,3);
var array2 = new Array("a","b","c");

var array3 = array1.concat( array2 );
alert( array3.length );
```

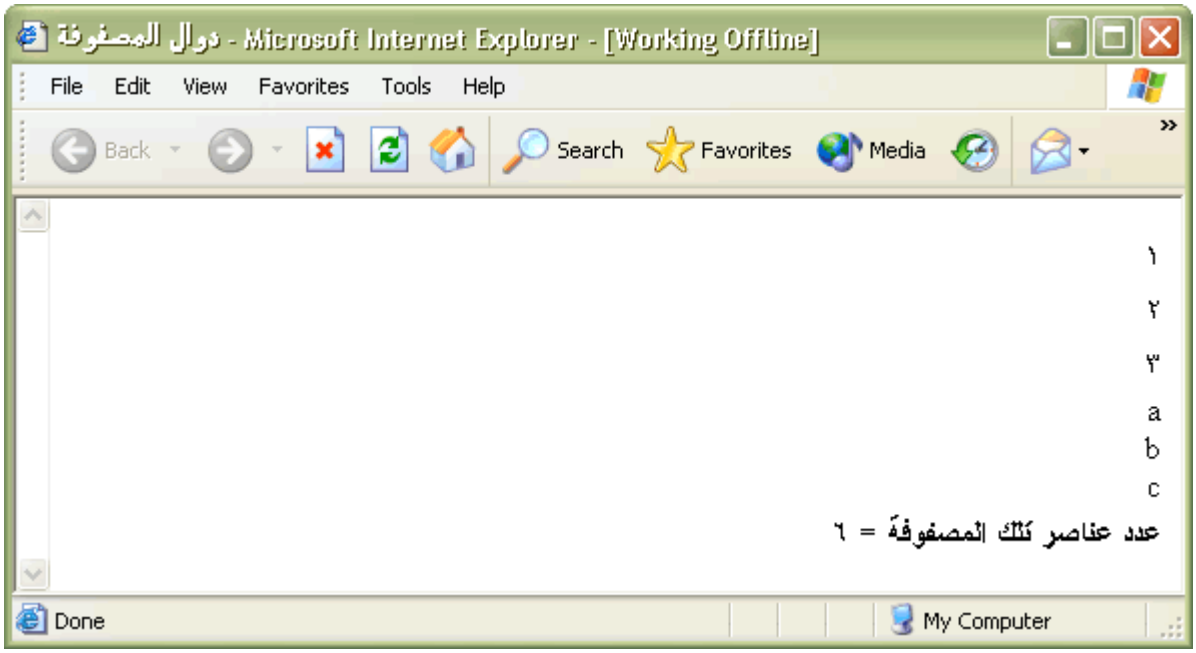
ويكون الناتج كما يلي

مثال

```
<HTML dir=rtl>
  <Title> دوال المصفوفة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var array1 = new Array(1,2,3);
    var array2 = new Array("a","b","c");

    var array3 = array1.concat( array2 );

    for( var i=0; i<array3.length; i++){
         document.write( array3[i]);
         document.write( "<br>" );
    }
    document.write( "<b>  = عدد عناصر تلك المصفوفة </b>" + array3.length );
    //-->
    </SCRIPT>
```

```
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



الدالة join

تستخدم لدمج قيم عناصر المصفوفة ، ويتم الدمج بين عناصر المصفوفة مع وضع فاصل بين قيم العناصر ويتم تحديد هذا الفاصل .

الصيغة العامة

```
    Array.join( الجملة الفاصلة );
```

مثال

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

    var str = arr.join(",");
    alert( str );

    str = arr.join(" ");
    alert( str );

    str = arr.join("");
    alert( str );
```

كما نلاحظ فقد تم دمج عناصر المصفوفة arr بوضع العلامة الفاصلة "،"

ويكون الناتج كما يلي

الدوال pop و push و shift و unshift

الدالة push : تستخدم لإضافة عنصر جديد في أخر المصفوفة
الدالة pop : تستخدم لحذف أخر عنصر من المصفوفة

الدالة unshift : تستخدم لإضافة عنصر في أول المصفوفة
الدالة shift : تستخدم لحذف أول عنصر من المصفوفة

الصيغة العامة

```
Array.push( قيمة العنصر الجديد );
Array.pop();

Array.unshift( قيمة العنصر الجديد );
Array.shift();
```

الدالة push : تستخدم لإضافة عنصر جديد في أخر المصفوفة

مثال

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

arr.push( "d" );

alert( arr.length );
alert( arr[3] );
```

ويكون الناتج كما يلي

الدالة pop : تستخدم لحذف أخر عنصر من المصفوفة

مثال

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

arr.pop();

alert( arr.length );
alert( arr[arr.length-1] );     // عرض أخر عنصر بالمصفوفة
```

ويكون الناتج كما يلي

الدالة shift : تستخدم لحذف أول عنصر من المصفوفة

مثال

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

arr.shift();

alert( arr.length );
alert( arr[0] );     // عرض أول عنصر بالمصفوفة
```

ويكون الناتج كما يلي

الدالة unshift : تستخدم لإضافة عنصر في أول المصفوفة

مثال

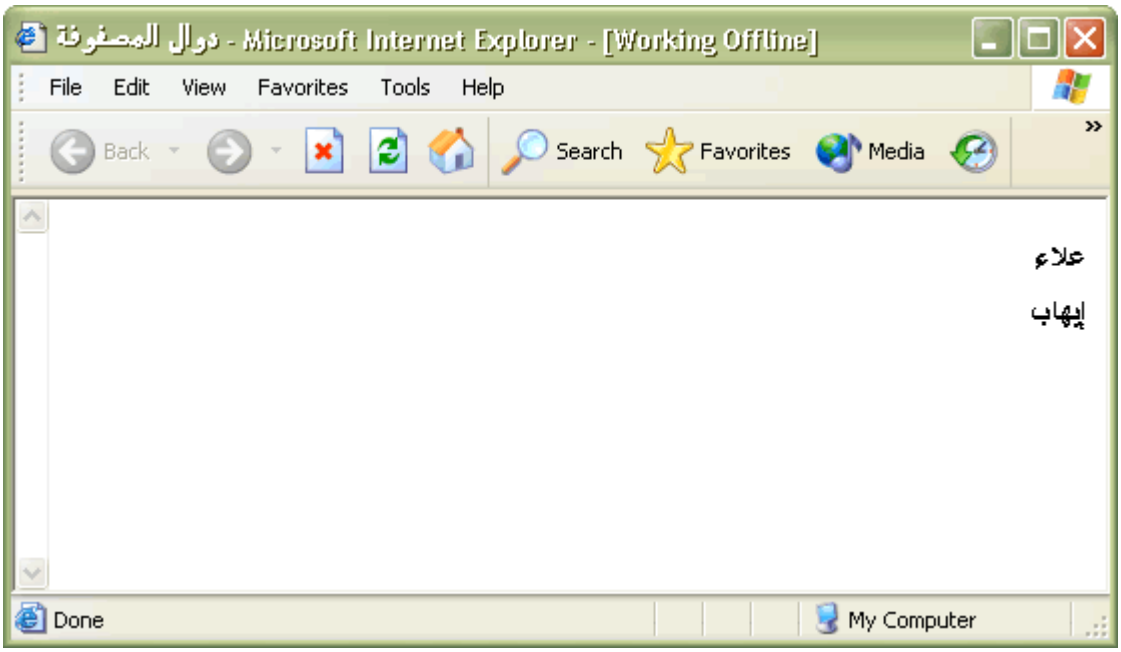
```
var arr = new Array("a","b","c");

arr.unshift( "x" );

alert( arr.length );
alert( arr[0] );     // عرض أول عنصر بالمصفوفة
```

ويكون الناتج كما يلي

مثال

```
<HTML dir=rtl>
  <Title> دوال المصفوفة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var arr    = new Array("إيهاب","محمد","علاء","حسين");
    var stack  = new Array();

    stack.push( arr[0] );
    stack.push( arr[2] );

    var str = stack.pop();
    document.write( "<b>" + str + "</b><br>");

    str = stack.pop();
    document.write( "<b>" + str + "</b>");
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي :



الدالة reverse

تستخدم لعكس مواقع عناصر المصفوفة .

الصيغة العامة

```
Array.reverse();
```

مثال

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

var arrReverse = arr.reverse();

alert( arrReverse.join(",") );
```

ويكون الناتج كما يلي

الدالة slice و splice

تستخدم كلتا الدالتان لإختزال عناصر المصفوفة .

الصيغة العامة للدالة slice

```
Array.slice( موقع البداية );
```

أو

```
Array.slice( موقع البداية , موقع أخر عنصر – ١ );
```

الصيغة العامة للدالة splice

```
Array.splice( موقع البداية , عدد العناصر );
```

مثال

```
var arr1 = new Array("a" ,"b" ,"c" ,"d" ,"e" ,"f");

var arr2 = arr1.slice(2);
alert( arr2.join(",") );
```

ويكون الناتج كما يلي

مثال

```
var arr1 = new Array("a" ,"b" ,"c" ,"d" ,"e" ,"f");

var arr2 = arr1.slice(2,4);
alert( arr2.join(",") );

var arr3 = arr1.splice(2,2);
alert( arr3.join(",") );
```

ويكون الناتج كما يلي

الدالة sort

تستخدم لترتيب عناصر المصفوفة .

الصيغة العامة

```
Array.sort();
```

أو

```
Array.sort( الدالة الخاصة بالترتيب );
```

مثال

```
var arr1 = new Array("c", "a", "b");

var arr2 = arr1.sort();

alert( arr2.join(",") );
```

ويكون الناتج كما يلي

لاحظ المثال التالي

```
var arr1 = new Array(12, 5, 200, 80);

var arr2 = arr1.sort();

alert( arr2.join(",") );
```

ويكون الناتج كما يلي

كما تري لم يتم الترتيب بشكل صحيح ، لأنه تم ترتيب عناصر قيم المصفوفة علي أساس أنها قيم نصية
ولحل هذة المشكلة يتم عمل دالة خاصة للترتيب كما يلي

مثال

```
<HTML dir=rtl>
  <Title> دوال المصفوفة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
```

```
    function compare(a, b) {
        return a - b;
    }

    var arr1 = new Array(12, 5, 200, 80);

    var arr2 = arr1.sort( compare );
    alert( arr2.join(",") );
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كما يلي

الدالة toLocaleString و toString

تقوم كلتا الدالتين بدمج قيم عناصر المصفوفة في متغير نصي بوضع فاصل بين القيم
فعند إستخدام الدالة toString يكون الفاصل المستخدم هو "،"
وبذلك فإنها تكافئ عمل الدالة join كما يلي

```
    Array.join(",");
```

أما عند إستخدام الدالة toLocaleString يكون الفاصل المستخدم هو ";"
وبذلك فإنها تكافئ عمل الدالة join كما يلي

```
    Array.join(";");
```

الصيغة العامة

```
Array.toLocaleString();
Array.toString();
```

مثال

```
var arr1 = new Array("a", "b", "c");

var str = arr1.toLocaleString();
alert( str );

str = arr1.toString();
alert( str );
```

ويكون الناتج كما يلي

الفصل الثالث

التعامل مع النصوص

سوف نناقش إن شاء الله في هذا الفصل النقاط التالية :

- النصوص البسيطة Simple String
- دمج النصوص Concatenation
- الدالة parseInt
- الدالة isNaN
- الدالة parseFloat
- علاج تداخل علامات التنصيص
- علامات الهروب أو العلامات الخاصة للكتابة Escaping Characters
- دوال تشفير وفك تشفير عناوين الأنترنت URL String Encoding and Decoding
    - الدالة escape
    - الدالة unescape

كما كنا قد أشرنا سابقا بالجزء الأول من الكتاب في فصل المتغيرات أن لغة الجافا سكربت ليست من اللغات التي تميز أنواع البيانات المخزنة بالمتغيرات يشكل قوي "not strongly typed language" يجعلها تميز بين المتغيرات الحرفية والمتغيرات النصية بشكل جيد ، كما سنري ظهور بعض المشاكل عند إستخدام المتغيرات النصية البسيطة لذلك ظهرت الحاجة لإنشاء كائن نصي يكون مبني داخل لغة الجافا سكربت يوفر أسلوب تعامل صحيح مع المتغيرات النصية

النصوص البسيطة Simple String

ما هو النص البسيط : يتكون النص من حرف أو أكثر وربما من رقم أو أكثر أو ربما يكون خليط من حروف مع أرقام لكن بشرط أن يتم وضع هذه الحروف بين علامتين تنصيص كما يلي

```
var str = "مرحبا بكم";
```

أو بين علامتين تنصيص فردي كما يلي

```
var str = 'مرحبا بكم';
```

لاحظ أيضا أن المتغير النصي ربما لا يحتوي علي اي حروف أو أرقام وهو ما يسمي بالمتغير النصي الفارغ Empty String كما يلي

```
var str = "";
```

دمج النصوص Concatenation

تعتبر عملية دمج النصوص من العمليات الهامة عند القيام بإنشاء صفحات للويب
ويتم دمج النصوص بإستخدام المعامل + أو المعامل =+ كما يلي

```
var str = "";

str += "<html><head><title>رأس الصفحة</title></head>";
str += "<body><h1>النصوص البسيطة</h1></body>";
str += "</html>";
```

مثال

```
var str1 = "أنت هنا معنا";
var str2 = "بالجزء الثاني";

var str3 = str1 + " " + str2;
alert( str3 );
```

ويكون الناتج كما يلي

مثال

```
var str1 = "2";
var str2 = 3;

var str3 = str1 + str2;
alert( str3 );
```

ويكون الناتج كما يلي

لاحظ معي أنه تم إعتبار قيمة المتغير str2 قيمة نصية بالرغم من أنها لم يتم وضعها بين علامتين تنصيص وكان المتوقع أن يكون ناتج الجمع هو ٥ .
ولعلاج هذة المشكلة يتم إستخدام الدالة parseInt أو الدالة parseFloat كما بالمثال التالي

مثال

في هذا المثال نطلب من المستخدم إدخال رقمين ، ثم نقوم بإظهار حاصل جمعهم

```
<HTML>
  <Title> النصوص البسيطة </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    var num1 = prompt("ادخل الرقم الأول","");
    var num2 = prompt("ادخل الرقم الثاني","");
```

```
   var num3 = parseInt(num1) + parseInt(num2);

    alert( num1 + " + " + num2 + " = " + num3 );
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج عند إدخال الرقم الأول بالقيمة ٤ والرقم الثاني بالقيمة ٨ ، فيكون ناتج الجمع هو ١٢ كما يلي

الدالة parseInt

تستخدم لتحويل القيم النصية إلي قيم عددية صحيحة (اي أنها تتعامل مع الأعداد الصحيحة وليست مع الأعداد النسبية) .

كما رأينا بالمثال السابق أننا قمنا بتحويل القيمة النصية الراجعة من الدالة prompt إلي قيمة عددية يمكننا إستخدامها في العمليات الحسابية بشكل صحيح

الصيغة العامة

```
    parseInt( القيمة النصية المراد تحويلها );
```

القيمة الراجعة من هذه الدالة : بعد عملية التحويل يتم إرجاع قيمة عددية صحيحة

مثال

```
    var num = parseInt( "23" );

    alert( num );
```

ويكون الناتج كما يلي

مثال

```
var num = parseInt( "23.43" );

alert( num );
```

ويكون الناتج كما يلي

لاحظ أن الدالة parseInt تقوم بالتحويل القيم النصية إلي قيم **عددية صحيحة** integer Number

لاحظ معي هذه الحالة
إذا قمنا بتمرير قيمة نصية للدالة ولكن هذه القيمة لا تمثل عدد كما يلي

```
parseInt( "h23" );
```

ويكون الناتج كما يلي

فقد قامت الدالة بإرجاع القيمة NaN وهي إختصار للجملة التالية Not a Numeric اي أن القيمة الممرة لها ليست قيمة عددية ولتفادي هذه الحالة يمكننا إستخدام الدالة isNaN .

الدالة isNaN

تستخدم للتأكد من حالة القيمة الممررة لها هل هي قيمة عددية أم لا
وكلمة isNaN هي إختصار للجملة التالية is Not a Number اي هل القيمة الممررة ليست قيمة عددية

الصيغة العامة

```
isNaN( القيمة );
```

القيمة الراجعة من هذه الدالة : تقوم بإرجاع قيمة من النوع البوليني اي إحدي القيمتين true أو false

مثال

```
alert( isNaN("1.3") );
```

ويكون الناتج كما يلي

الدالة parseFloat

تستخدم لتحويل القيم النصية إلي قيم عددية نسبية (اي أنها تتعامل مع الأعداد الصحيحة و الأعداد النسبية) .

الصيغة العامة

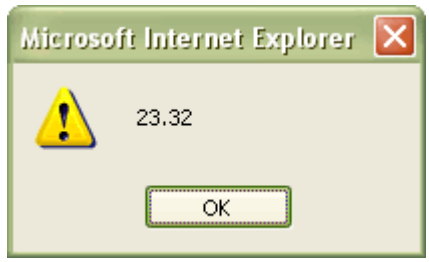
```
parseFloat( القيمة النصية المراد تحويلها );
```

القيمة الراجعة من هذه الدالة : بعد عملية التحويل يتم إرجاع قيمة عددية نسبية

مثال

```
var num = parseFloat( "23.32" );

alert( num );
```

ويكون الناتج كما يلي



علاج تداخل علامات التنصيص

كما ذكرنا سابقا أن النص هو عبارات وجمل يتم كتابتها بين علامتين تنصيص ""
علي سبيل المثال نريد عمل نص به عبارة أنا من محبي لغة الجافا سكربت سوف نقوم بوضعها بين علامتين التنصيص كما يلي " **أنا من محبي لغة الجافا سكربت** " وبذلك يستطيع مفسر اللغة تميز أن هذه الكلمات تابعة لنص واحد .

إذا إين المشكلة ومتي تظهر نعم أنت علي حق لا توجد مشكلة هنا إلا إذا حدث التالي
تخيل معي أن نص الجملة يحتوي علامة تنصيص أو أكثر
علي سبيل المثال أنك تريد كتابة الجملة التالية (**هل هناك "مشكلة" يا رجل ؟**) كما تري أننا الأن في مأزق لأننا لو وضعنا هذة الجملة بين علامتيين تنصيص سوف يحدث تداخل في علامات التنصيص وسوف يؤدي هذا إلي إرتباك لمفسر اللغة مما ينتج عنه خطأ لغوي syntax error .
الأن ما الحل ؟
ربما يتبادر إلي ذهنك الهرب من المشكلة وتقول أنا لست في حاجة لإظهار علامات التنصيص في الجملة وسوف أجعل الجملة بدونهما كما هو حالنا نحن العرب ولكن دائما تأتي الحلول لتفادي الأخطاء وتجنبها وليس الهرب منها ، أعتقد أنه تبادر إليك الأن أن هذه المشكلة لها حل ابشرك بقولي نعم
حيث توفر لنا لغة الجافا سكربت علامات الهروب Escaping Characters .

علامات الهروب أو العلامات الخاصة للكتابة Escaping Characters

تمكنك من تضمين بعض الحروف التي يصعب كتابتها في محتوي النصوص ومنها التالي :

\'	تمكنك من إضافة علامه التنصيص الفردية بداخل النص
\"	تمكنك من إضافة علامه التنصيص بداخل النص
\r	تمكنك من إدخال حرف خاص لعمل ما يسمي Carriage return اي تراجع موشر الكتابه إلي بداية السطر مما ينتج عنه في بعض الأحيان عمل سطر جديد
\n	تمكنك من عمل سطر جديد داخل النص
\t	تمكنك من إدخال الحرف Tab

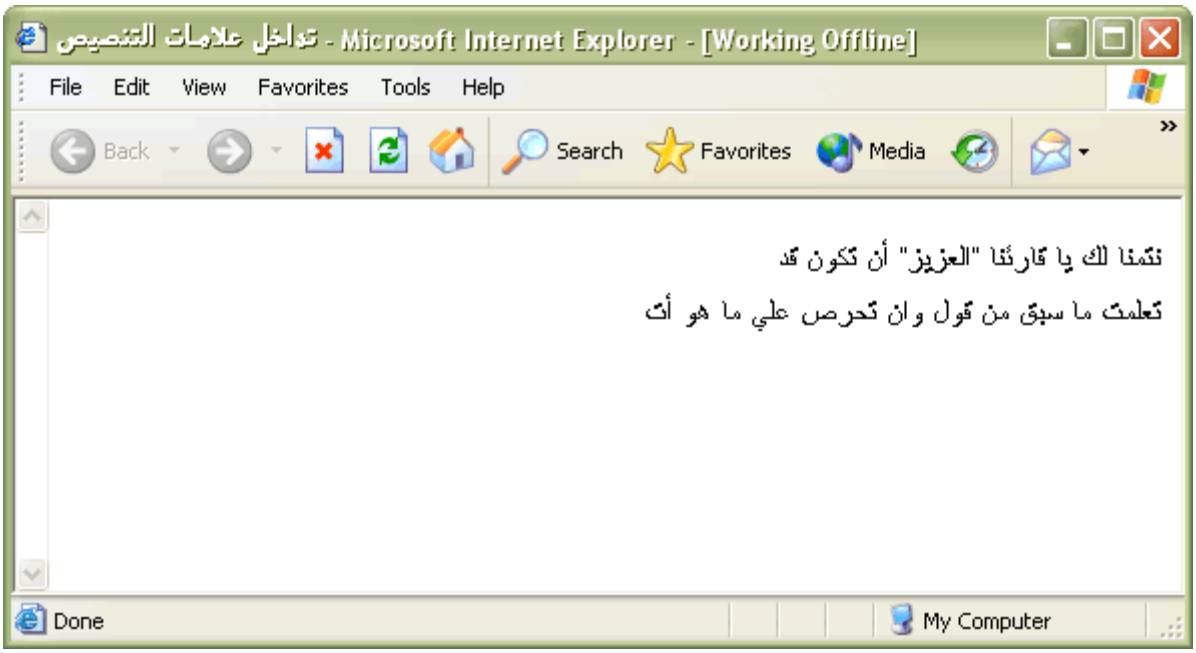
بعد التعرف السريع لعلامات الهروب كيف يمكننا حل مشكلة كتابة النص السابق (هل هناك "مشكلة" يا رجل ؟)
بكل يسر يتم إستبدال اي علامة تنصيص (") بعلامة الهروب ("\) ويكون النص كالتالي
(هل هناك \"مشكلة\" يا رجل ؟)

مثال تطبيقي لكتابة الرسالة التالية :

نتمنا لك يا قارئنا "العزيز" أن تكون قد
تعلمت ما سبق من قول وان تحرص علي ما هو أت

```
<HTML dir=rtl>
  <Title> تداخل علامات التنصيص </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
            document.write("نتمنا لك يا قارئنا \"العزيز\" أن تكون قد " + "<br>" +
                        "تعلمت ما سبق من قول وان تحرص علي ما هو أت");
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج كالتالي



دوال تشفير وفك تشفير عناوين الأنترنت URL String Encoding and Decoding

أحيانا عندما نقوم بإرسال عنوان لاحدي المواقع ، بأن نقوم علي سبيل المثال بكتابة العنوان التالي "http://www.islamway.com/index.php?action=all news" في شريط العنوان بالمتصفح Address Bar فعند الضغط علي زرار ذهاب go يتم تشفير عنوان الموقع url تلقائيا اي يتم تحويل العلامات الخاصة إلي ما يقابلها في النظام الستعشري بنظام التشفير الأسكي ASCII ويكون العنوان الناتج كما يلي "http://www.islamway.com/index.php?action=all%20news " لاحظ أن حرف المسافة بين الكلمتين all و الكلمة news تم إستبدالها بالرمز التالي "%20" والرقم ٢٠ هو الرقم الستعشري في نظام التشفير الأسكي المقابل لحرف المسافة .

لكن أحيانا نقوم نحن بإستخدام أحدي الدوال لعمل إرسال لعنوان موقع من خلال كود الجافا سكربت وبذلك فنحن في عرضة لعدم تشفير بيانات عنوان الموقع المرسل لذلك نحتاج إلي كلتا الدالتين escape و unescape .

الدالة escape

تستخدم لتشفير العلامات الخاصة الموجودة بعنوان الموقع url أو اي نص (ليس بالضرورة أن يكون عنوان لموقع) إلي ما يقابلها في النظام الستعشري بنظام التشفير الأسكي ASCII مضافا إلية العلامة % .

الصيغة العامة

```
escape( عنوان الموقع المراد تشفيره );
```

القيمة الراجعة من هذه الدالة : بعد عملية التحويل يتم إرجاع قيمة نصية

مثال

```
var siteURL = "http://www.islamway.com/index.php?action=all news";
var codedSiteURL = escape( siteURL );

alert( codedSiteURL );
```

ويكون الناتج كالتالي

وكما تري فقد تغير عنوان الموقع إلي عنوان خطأ لأنه تم تشفير كل العنوان ، ولكننا دائما ما نحتاج إلي تشفير البيانات المرسلة فقط كما يلي

مثال

```
var siteURL = "http://www.islamway.com/index.php?action=";
var codedSiteURL = siteURL + escape( "all news" );

alert( codedSiteURL );
```

ويكون الناتج كالتالي

الدالة unescape

تستخدم لفك تشفير العلامات الخاصة الموجودة بعنوان الموقع url أو اي نص (ليس بالضرورة أن يكون عنوان لموقع) إي عكس ما تقوم به الدالة escape .

الصيغة العامة

```
unescape( عنوان الموقع المراد فك تشفيره );
```

القيمة الراجعة من هذه الدالة : بعد عملية التحويل يتم إرجاع قيمة نصية

مثال

```
// تشفير النص
var siteURL = "http://www.islamway.com/index.php?action=";
var codedSiteURL = siteURL + escape( "all news" );

// فك تشفير النص
var UnCodedSiteURL = unescape(codedSiteURL);

alert( UnCodedSiteURL );
```

ويكون الناتج كالتالي

الفصل الرابع

كائن النصوص

سوف نناقش إن شاء الله في هذا الفصل النقاط التالية :

- كائن النصوص String Object
- إنشاء كائن نصي
- خصائص كائن النصوص String Object Properties
- دوال كائن النصوص String Object Methods
  - الدوال الخاصة بالتعامل مع النصوص parsing Methods
  - الدوال الخاصة بتنسيق النص Formatting Methods

كائن النصوص String Object

إنشاء كائن نصي

يمكننا إنشاء الكائن النصي بعدة طرق كما يلي

١- **إنشاء كائن نصي بإستخدام الكائن** String

```
var str = new String();
```

أو

```
var str = new String("النص");
```

٢- **لاحظ أن النص البسيط يمكن إستخدامه مثل المتغير الناشئ من الكائن** String

```
var literalStr = "مرحبا بالنصوص";
```

يوفر لنا الكائن النصي العديد من الخصائص والدوال المهمة ، لذلك دعنا نتحدث الأن عن الخصائص و الدوال الخاصة بكائن النصوص

خصائص كائن النصوص String Object Properties

الخاصية constructor

سوف نتحدث عن الخاصية constructor بالجزء الثالث من هذا الكتاب ، عند الحديث عن الكائنات والبرمجة الكائنية

الخاصية prototype

سوف نتحدث عن الخاصية prototype بالجزء الثالث من هذا الكتاب ، عند الحديث عن الكائنات والبرمجة الكائنية والوراثة Inheritance

الخاصية length

تستخدم الخاصية length لحساب طول النص اي عدد الحروف المكونة لهذا النص

مثال

```
var strObject = new String("الكائن النصي");

alert( strObject.length );
```

أو كما يلي

```
var strObject = new String();
strObject = "الكائن النصي";

alert( strObject.length );
```

ويكون الناتج كما يلي

مثال

```
var str = "المتغير النصي البسيط";

alert( str.length );
```

أو كما يلي

```
alert( "المتغير النصي البسيط".length );
```

ويكون الناتج كما يلي

دوال كائن النصوص String Object Methods

سوف نقسم دوال الكائن النصي إلي قسمين أحدهما للتعامل مع تقطيع النصوص والأخر لتغير نسق النص من لون وسمك ألخ عند كتابة تلك النص في المتصفح

الدوال الخاصة بالتعامل مع النصوص parsing Methods

الدالة charAt

تستخدم لقراءة حرف واحد في موقع معين من النص

القيمة الراجعة : تقوم بإرجاع قيمة نصية

الصيغة العامة

```
String.charAt( موقع الحرف );
```

مثال

```
var str = new String();
str = "لا اله إلا الله محمد رسول الله";

alert( " : الحرف الذي يقع عند الموقع الثاني" + str.charAt(1) );

var char = str.charAt(7);
alert( " : الحرف الذي يقع عند الموقع الثامن" + char );
```

ويكون الناتج كما يلي

تمرين

نريد عكس نص يتم إستقباله من المستخدم

```
<HTML>
  <Title> عكس النص </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    // الدالة الخاصة بعكس النص
    function reverseString( str ){
        var Revstr = "";
        for( var i = str.length-1; i>=0; i--)
             Revstr += str.charAt(i);

        return Revstr;
    }

    var strFromUser = prompt("ادخل النص","");
    alert( reverseString(strFromUser) );

    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

ويكون الناتج عند إدخال القيمة hello هو olleh كما يلي

الدوال charCodeAt و fromCharCode

الدالة charCodeAt

تستخدم لقراءة حرف واحد في موقع معين من النص ولكن تقوم بإرجاع القيمة العددية بنظام التشفير الأسكي ASCII لهذا الحرف

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
String.charCodeAt( موقع الحرف );
```

أو

```
String.charCodeAt();
```

مثال

```
"abc".charCodeAt();        // ينتج عنه القيمة : 97
"abc".charCodeAt(0);       // ينتج عنه القيمة : 97
"abc".charCodeAt(1);       // ينتج عنه القيمة : 98
```

الدالة fromCharCode

تستخدم لتحويل قيمة عددية أو أكثر بنظام الأسكي ASCII إلي ما يقابلها من حروف
اي أنها تقوم بعكس وظيفة الدالة charCodeAt

القيمة الراجعة : تقوم بإرجاع قيمة نصية مكونة من حرف أو أكثر

الصيغة العامة

```
String.fromCharCode ( القيمة العددية );
```

أو

```
String.fromCharCode( أكثر من قيمة عددية );
```

مثال

```
String.fromCharCode(97);                // ينتج عنه القيمة : a

String.fromCharCode(97, 98 ,99);        // ينتج عنه القيمة : abc
```

الدالة concat

تستخدم لدمج قيمة نصية بقيمة نصية أخري

القيمة الراجعة : تقوم بإرجاع قيمة نصية مكونة ناتج دمج نصين

الصيغة العامة

```
String.concat( القيمة النصية );
```

مثال

```
"abc".concat("def");                    // ينتج عنه القيمة : abcdef
```

الدوال indexOf و lastIndexOf

الدالة indexOf

تستخدم للبحث عن موقع نص داخل نص أخر ، **ويبدأ البحث من أول حرف بالنص** الذي يتم البحث فيه

القيمة الراجعة : تقوم بإرجاع قيمة عددية تعبر عن موقع النص الذي يتم البحث عنه داخل النص المبحوث فبه

الصيغة العامة

```
String.indexOf( القيمة النصية المراد البحث عنها );
```

أو

```
String.indexOf( القيمة النصية المراد البحث عنها , موقع بداية البحث);
```

مثال

```
"abc".indexOf("b");                     // ينتج عنه القيمة : 1
"abc".indexOf("c");                     // ينتج عنه القيمة : 2

"abc".indexOf("bd");                    // ينتج عنه القيمة : -1
"abc".indexOf("g");                     // ينتج عنه القيمة : -1

"abc".indexOf("b" ,1);                  // ينتج عنه القيمة : 1
"abc".indexOf("b" ,2);                  // ينتج عنه القيمة : -1
```

كما نري إذا لم يتم العثور علي النص يتم إرجاع القيمة - ١

تمرين

نريد التأكد من صحة قيمة بريد إليكتروني يتم إستقباله من المستخدم
مع العلم أن البريد الأليكتروني يتميز بوجود الرمز @

```
<HTML>
  <Title> التأكد من صحة البريد الإليكتروني </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    // الدالة الخاصة للتأكد من وجود الرمز @
    // بنص البريد الإليكتروني المرسل لها
    function IsValidEmail( Emailstr ){
        return ( Emailstr.indexOf("@") != -1 )? true : false ;
    }

    var EmailstrFromUser = prompt("ادخل بريدك الأليكتروني","");

    if( IsValidEmail(EmailstrFromUser) )
        alert(" هذا البريد الإليكتروني صحيح ");
    else
        alert( " هذا البريد الإليكتروني غير صحيح " );

    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

وسوف نري بالفصل القادم "التعامل مع التعبيرات المنتظمة" كيف يتم التأكد من البريد الإليكتروني بشكل أكثر دقة

الدالة lastIndexOf

تستخدم للبحث عن موقع نص داخل نص أخر ، **ويبدأ البحث من أخر حرف بالنص** الذي يتم البحث فيه ، وهي بذلك تتشابه إلي حد ما مع الدالة indexOf .

القيمة الراجعة : تقوم بإرجاع قيمة عددية تعبر عن موقع النص الذي يتم البحث عنه داخل النص المبحوث فبه

الصيغة العامة

```
    String.lastIndexOf( القيمة النصية المراد البحث عنها );
```

أو

```
    String.lastIndexOf( القيمة النصية المراد البحث عنها , موقع بداية البحث);
```

مثال

```
"abcdef".lastIndexOf("b");                    // ينتج عنه القيمة : 1
"abcdef".lastIndexOf("c");                    // ينتج عنه القيمة : 2

"abcdef".lastIndexOf("bd");                   // ينتج عنه القيمة : -1
"abcdef".lastIndexOf("g");                    // ينتج عنه القيمة : -1

"abcdef".lastIndexOf("b" ,1);                 // ينتج عنه القيمة : 1
"abcdef".lastIndexOf("b" ,2);                 // ينتج عنه القيمة : 1
```

الدوال match و replace و search

سوف يتم مناقشتهم في الفصل القادم "التعامل مع التعبيرات المنتظمة"

الدوال slice و substr و substring

الدالة slice و substring

تستخدم لإستخلاص جزء من النص بدون أن تؤثر علي النص الأساسي

القيمة الراجعة : تقوم بإرجاع قيمة نصية

الصيغة العامة

```
String.slice( موقع البداية );
String.substring( موقع البداية );
```

أو

```
String.slice( موقع البداية , موقع النهاية);
String.substring ( موقع البداية , موقع النهاية);
```

مثال

```
"abcdef".substring(1);                // ينتج عنه القيمة : bcdef
"abcdef".substring(1,3);              // ينتج عنه القيمة : bc
```

لاجظ التالي

مثال

```
"abcdef".substring(2, -1);                    // ينتج عنه القيمة : ab
```

لأنه عندما يكون موقع النهاية الممرر للدالة substring أو slice قيمة سالبة يتم البحث من نهاية موقع البداية فكما بالمثال السابق كان موقع البداية هو ٢ اي ما يكافئ الحرف c ثم تم البحث لقيمة موقع النهاية بقيمة سالبة تساوي – ١ اي أنه ستتم عملية إختزال النص في أتجاه اليسار إبتدأ من الحرف موقع الحرف b .

الدالة substr

تستخدم لإستخلاص جزء من النص بدون أن تؤثر علي النص الأساسي ، كما تفعل الدالة substring مع وجود فارق أن المعامل الثاني الممرر للدالة substr يعبر عن عدد الحروف المختزلة أو طول النص الناتج ، بينما يعبر عن موقع النهاية بالنسبة للدالة substring .

القيمة الراجعة : تقوم بإرجاع قيمة نصية

الصيغة العامة

```
String.substr( موقع البداية );
```

أو

```
String.substr ( موقع البداية , طول النص );
```

مثال

```
"abcdef".substr(1);                   // ينتج عنه القيمة : bcdef
"abcdef".substr(1,3);                 // ينتج عنه القيمة : bcd
```

تمرين

نريد إستخلاص اسم المستخدم وسم مزود خدمة البريد الإليكتروني لقيمة بريد إليكتروني يتم إستقباله من المستخدم

مع العلم أن اسم المستخدم يسبق موقع الرمز @ و اسم مزود الخدمة يلي الرمز @ فعلي سبيل المثال
إذا كان البريد الإليكتروني المدخل هو a_elhussein@hotmail.com
يكون اسم المستخدم a_elhussein
ويكون اسم المزود hotmail

```
<HTML dir=rtl>
  <Title> البريد الإليكتروني </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    var EmailstrFromUser;
    EmailstrFromUser = prompt("ادخل بريدك الأليكتروني","");

    var indexOfDelimiter = EmailstrFromUser.indexOf("@");
    if( indexOfDelimiter != -1 ){
        var username = EmailstrFromUser.substring(0, indexOfDelimiter);

        var indexOfDot = EmailstrFromUser.indexOf(".");
        var serverName = EmailstrFromUser.substring( indexOfDelimiter+1, indexOfDot);

        alert( " : اسم المستخدم" + username + "\r\n" +
               " : اسم المزود" + serverName );
    }else{
        alert( " هذا البريد الإليكتروني غير صحيح " );
    }

    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

الدالة split

تستخدم لتقطيع النص إستنادا علي جملة التقطيع الممررة لها ، ثم تقوم بوضع الجمل المقطعة في مصفوفة ذات بعد واحد .

القيمة الراجعة : تقوم بإرجاع مصفوفة تحمل الجمل المقطعة

الصيغة العامة

```
String.split( الجملة المراد التقطيع بها );
```

أو

```
String.split( الجملة المراد التقطيع بها , طول المصفوفة الناتج);
```

مثال

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

ينتج عنها المصفوفة التالية

```
Arr[0] ----------à ab
Arr[1] ----------à ef
```

مثال

```
var arr = "abcdef".split("cd");

alert( arr.toString() );
```

ويكون الناتج كما يلي

مثال

```
var arr = "abcdef".split("cd", 1);

alert( arr.toString() );
```

ويكون الناتج كما يلي

الدالة toLowerCase و toUpperCase

تستخدم الدالة toLowerCase لتحويل حالة الحروف بالنص إلي حروف صغيرة small letters
وعلي العكس تستخدم الدالة toUpperCase لتحويل حالة الحروف بالنص إلي حروف كبيرة capital letters .

القيمة الراجعة : تقوم بإرجاع قيمة نصية

الصيغة العامة

```
String.toLowerCase();
String.toUpperCase();
```

مثال

```
"abcdef".toUpperCase();    // ينتج عنه القيمة : ABCDEF
"AbcDef".toLowerCase();    // ينتج عنه القيمة : abcdef
```

الدالة toString و valueOf

تستخدم لعرض قيمة النص

القيمة الراجعة : تقوم بإرجاع قيمة نصية

الصيغة العامة

```
String.toString);
String.valueOf();
```

مثال

```
"abcdef".toString();       // ينتج عنه القيمة : abcdef
"abcdef".valueOf();        // ينتج عنه القيمة : abcdef
```

الدوال الخاصة بتنسيق النص Formatting Methods

تستخدم لتغير نسق النص

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

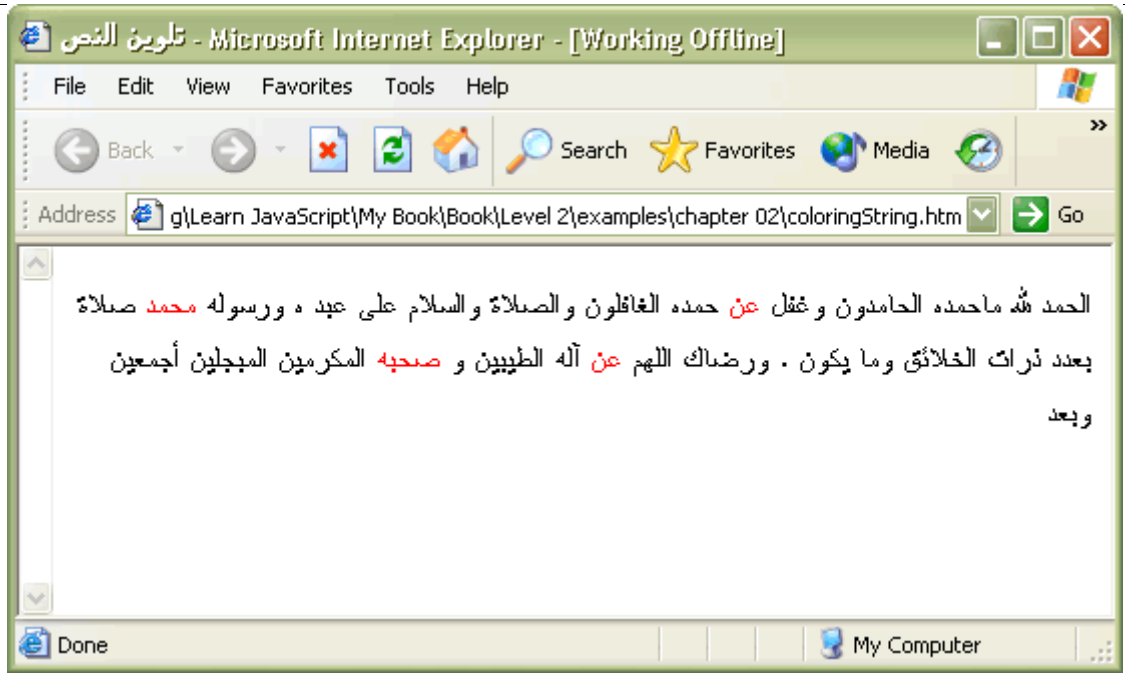
تمرين

نريد إستخلاص الكلمات التالية "محمد" و "صحبه" و "عن" من النص التالي

"الحمد لله ماحمده الحامدون وغفل عن حمده الغافلون والصلاة والسلام على عبد ه ورسوله محمد صلاة بعدد ذرات الخلائق وما يكون . ورضاك اللهم عن آله الطيبين وصحبه المكرمين المبجلين أجمعين وبعد" وتلوينها باللون

الأحمر

```
<HTML dir=rtl>
  <Title> تلوين النص </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    var str = " الحمد لله ماحمده الحامدون وغفل عن حمده الغافلون  والصلاة والسلام على عبد ه ورسوله
محمد صلاة بعدد ذرات الخلائق وما يكون . ورضاك اللهم عن آله الطيبين و صحبه المكرمين المبجلين أجمعين وبعد";

    var wordsArray = str.split(" ");
    var resultStr = "";
    for(var i=0; i< wordsArray.length; i++){
        var word = wordsArray[i];

        if(word == "محمد" || word == "صحبه" || word == "عن")
            resultStr += word.fontcolor("red");
        else
            resultStr += word;

        resultStr += " ";
    }

    document.write(resultStr);
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```


تلوين النص - Microsoft Internet Explorer - [Working Offline]
File Edit View Favorites Tools Help
Back Search Favorites Media
Address g\Learn JavaScript\My Book\Book\Level 2\examples\chapter 02\coloringString.htm Go
الحمد لله ماحمده الحامدون وغفل عن حمده الغافلون والصلاة والسلام على عبده ورسوله محمد صلاة
بعدد ذرات الخلائق وما يكون . ورضاك اللهم عن آله الطيبين و صحبه المكرمين المبجلين أجمعين
وبعد
Done My Computer

الفصل الخامس

التعامل مع التعبيرات المنتظمة

Regular Expressions

سوف نناقش إن شاء الله في هذا الفصل النقاط التالية :

- ما هي التعبيرات المنتظمة Regular Expressions
- طريقة كتابة التعبيرات المنتظمة
  - الأنماط patterns
  - المجموعات Grouping
- إستخدام كائن النصوص مع الأنماط
- كائن التعبيرات المنتظمة RegExp

ما هي التعبيرات المنتظمة Regular Expressions

يمكننا القول بأن التعبيرات المنتظمة هو أسلوب متقدم للتعامل مع النصوص ، لتوفير الوقت والمجهود المبذول عند التعامل مع النصوص كما رأينا بالفصل الثاني الخاص بالتعامل مع النصوص من تقطيع و معاينة موقع جملة معينة بنص ما .

ولمن يبرمج بلغة مثل لغة perl البيرل ، فلن يجد إختلاف في طريقة إستخدام التعبيرات الخاصة بلغة الجافا سكربت عما تعلمه بلغة البيرل .

توفر أيضا التعبيرات الخاصة أسلوب قوي للتعامل مع النصوص ، مع القدرة العالية لتقطيع النصوص وإستبدال نص بنص أخر وكثير من المهام الأخري .

طريقة كتابة التعبيرات المنتظمة Regular Expressions

يمكننا كتابة التعبيرات المنتظمة بناء علي ما يسمي بالأنماط patterns و عمل المجموعات grouping وسوف نتحدث عن كل عنصر علي حدي

الأنماط patterns

تنقسم الأنماط إلي نوعين أنماط بسيطة و أخري مركبة

الأنماط البسيطة

أمثلة علي الأنماط البسيطة

```
var pattern = / /;
```

فهذا النمط يعبر عن الحرف "مسافة" وفيما بعد سوف نري كيفية إستخدام هذا النمط لإجراء عمليات البحث و الإستبدال

```
var pattern = /hello/;
```

فهذا النمط يعبر عن الكلمة "hello"

```
var pattern = /hello patterns/;
```

فهذا النمط يعبر عن الجملة "hello patterns"

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

لاحظ وجود الحرف i الذي يجعل عمليات البحث والإستبدال القائمة علي هذا النمط لا تتأثر بحالة الحروف كبيرة أم صغيرة ، وحرف i هو إختصار لي Case Insensitive

لذلك هذا النمط يعبر عن كلمة "WEB" أو "web" أو "Web" أو "weB" وهكذا .

```
var pattern = /web/g;
```

لاحظ وجود الحرف g الذي يجعل عمليات البحث والإستبدال القائمة علي هذا النمط تتم علي كل النص ولا تتوقف بمجرد أن يتم توافق النمط مع جزء من النص ، وحرف g هو إختصار لي global

لذلك هذا النمط يعبر عن كلمة "web" فقط ولكن يتم البحث عنها في كل النص بدون توقف .

ويمكننا دمج الرمزين i و g كما يلي

```
var pattern = /web/gi;
```

الأنماط المركبة

تمكننا الأنماط المركبة من إجراء تعبيرات أكثر قوة ، ومثل علي ذلك تمكننا الأنماط المركبة من عمل التالي

إستخدام الرموز الخاصة بالتكرار

الرمز	الوصف
.	تعبر عن حرف واحد
?	تعبر عن وجود حرف واحد علي الأكثر ، اي ربما تعبر عن عدم وجود اي حرف
*	تعبر عن وجود حرف أو أكثر أو لا شئ
+	تعبر عن وجود حرف أو أكثر

مثال

```
var pattern = /w.eb/;
```

لاحظ وجود الحرف "." داخل نص النمط ، لذلك هذا النمط يعبر عن كلمة "waeb" أو "Wbeb" وهكذا ولا يعبر عن "web" لأن الحرف "." يعبر عن وجود حرف واحد بدون تحديد لهذا الحرف.

مثال

```
var pattern = /c.t/;
```

هذا النمط يمكن أن يعبر عن "cat" أو "cut" وهكذا ولا يعبر عن "ct" ولا عن "category"

مثال

```
var pattern = /w?eb/;
```

لاحظ وجود الحرف "?" داخل نص النمط ، لذلك هذا النمط يعبر عن كلمة "waeb" أو "Wbeb" وهكذا و أيضا يعبر عن "web" لأن الحرف "?" يعبر عن وجود حرف واحد بدون تحديد لهذا الحرف وربما يعبر عن لا شئ.

مثال

```
var pattern = /C*t/;
```

هذا النمط يمكن أن يعبر عن "cat" و "cut" و "ct" و "coot" ولا يعبر عن "category"

مثال

```
var pattern = /C*t*/;
```

هذا النمط يمكن أن يعبر عن "cat" و "cut" و "ct" و "coot" و "category"

مثال

```
var pattern = /C+t/;
```

هذا النمط يمكن أن يعبر عن "cat" و "cut" و "coot" ولا يعبر عن "ct" أو "category"

إستخدام رموز البداية والنهاية الخاصة

الرمز	الوصف
^	تعبر عن أن عند إجراء عملية تماثل النص مع النمط يجب أن يتكافئ النمط مع بداية النص
$	تعبر عن أن عند إجراء عملية تماثل النص مع النمط يجب أن يتكافئ النمط مع نهاية النص

مثال

```
var pattern = /^web/;
```

لاحظ وجود الحرف " ^ " بالنمط ، لذلك هذا النمط يمكن أن يعبر عن "web" ولا يعبر عن "this is a web" لأننا أشترطنا في هذا النمط أن تأتي كلمة web في أول النص .

مثال

```
var pattern = /web$/;
```

لاحظ وجود الحرف "$" بالنمط ، لذلك هذا النمط يمكن أن يعبر عن "web" أو "this is a web" لأننا أشترطنا في هذا النمط أن تأتي كلمة web في أخر النص ، لذلك هذا النمط لا يعبر عن "is web , good".

مثال

```
var pattern = /^web$/;
```

لاحظ وجود الحرفين " ^ " **و** "$" بالنمط ، لذلك هذا النمط يمكن أن يعبر عن "web" فقط .

إستخدام رموز التكرار العددية

الرمز	الوصف
{n}	تعبر عن أن يجب تكرار الحرف السابق له عدد من المرات يساوي n
{n,m}	تعبر عن أن يجب تكرار الحرف السابق له عدد من المرات يساوي من القيمة n إلي القيمة m
{n,}	تعبر عن أن يجب تكرار الحرف السابق له عدد من المرات يساوي من القيمة n إلي مالانهاية

مثال

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

هذا النمط يمكن أن يعبر عن "cct" ولا يعبر عن "cat" ولا عن "ct"

مثال

```
var pattern = /c{1,2}t/;
```

هذا النمط يمكن أن يعبر عن "cct" أو "ct" ولا يعبر عن "cat"

مثال

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

هذا النمط يمكن أن يعبر عن "cct" أو "cccccct" ولا يعبر عن "cat" ولا عن "ct"

مثال

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

هذا النمط يمكن أن يعبر عن "cotton" أو "cutter"

لاحظ الحالات التالية

الحالة الأولي

```
var pattern = /a?/;
```

يكافئ

```
var pattern = /a{0,1}/;
```

الحالة الثانية

```
var pattern = /a./;
```

يكافئ

```
var pattern = /a{1}/;
```

الحالة الثالثة

```
var pattern = /a*/;
```

يكافئ

```
var pattern = /a{0,}/;
```

الحالة الرابعة

```
var pattern = /a+/;
```

يكافئ

```
var pattern = /a{1,}/;
```

إستخدام رموز المدي range

الرمز	الوصف
[ ]	تعبر عن أنه يتم البحث في المدي المحدد
[^]	تعبر عن أنه يتم البحث في عكس المدي المحدد

مثال

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

هذا النمط يمكن أن يعبر عن اي حرف صغير small letter يقع بين الحرفين a و z

مثال

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

هذا النمط يمكن أن يعبر عن "hello" ولا يعبر عن "HELLO"

مثال

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

هذا النمط يمكن أن يعبر عن "HELLO" ولا يعبر عن "hello"

بعض الأنماط المفيدة

النمط التالي يعبر عن جميع الحروف التي يمكن كتابتها بجميع الحالات (صغيرة أو كبيرة)

```
var pattern = /[a-zA-Z]/;
```

النمط التالي لا يعبر عن جميع الحروف التي يمكن كتابتها بجميع الحالات (صغيرة أو كبيرة)

```
var pattern = /[^a-zA-Z]/;
```

النمط التالي يعبر عن جميع الأعداد من صفر إلي ٩

```
var pattern = /[0-9]/;
```

النمط التالي لا يعبر عن جميع الأعداد من صفر إلي ٩

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

النمط التالي يعبر عن جميع الأعداد من صفر إلي ٩ وجميع الحروف المكتوبة

```
var pattern = /[a-zA-Z0-9]/;
```

مثال

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

هذا النمط يمكن أن يعبر عن "2web"

مثال

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

هذا النمط يمكن أن يعبر عن "aweb" ولا يعبر عن "2web"

إستخدام الرموز الخاصة

الرمز	الوصف
\d	تعبر عن الأرقام من صفر إلي ٩ اي أنها تكافئ النمط التالي [0-9]
\D	لا تعبر عن الأرقام من صفر إلي ٩ اي أنها تكافئ النمط التالي [^0-9]
\w	تعبر عن جميع الرموز التي يمكن كتابتها كالأرقام من صفر إلي ٩ أو عن الحروف من a إلي z أو عن الحرف _ أو ما يسمي بالأندرسكور underscore اي أنها تكافئ النمط التالي [a-zA-Z0-9_]
\W	لا تعبر عن جميع الرموز التي يمكن كتابتها كالأرقام من صفر إلي ٩ أو عن الحروف من a إلي z أو عن الحرف _ أو ما يسمي بالأندرسكور underscore

	اي أنها تكافئ النمط التالي [^a-zA-Z0-9_]
\s	تعبر عن جميع الرموز الخاصة التي يصعب كتابه بعضها مثل حرف المسافة والتاب tab والسطر الجديد \n والرموز التالية \r و \f
\S	لا تعبر عن جميع الرموز الخاصة التي يصعب كتابه بعضها مثل حرف المسافة والتاب tab والسطر الجديد \n والرموز التالية \r و \f
\b	تعبر عن أن النمط يجب أن يتواجد في أول الكلمة فقط علي سبيل المثال النمط /\bor/ يعبر عن organ ولا يعبر عن normal والنمط /or\b/ يعبر عن traitor ولا يعبر عن perform
\B	لا تعبر عن أن النمط يجب أن يتواجد في أول الكلمة فقط علي سبيل المثال النمط /\Bor/ لا يعبر عن organ ولكن يعبر عن normal

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

المجموعات Grouping

تستخدم لتجميع نواتج البحث في مجموعات يتم تخزينها في مصفوفة أو في متغيرات تبدأ بالرمز $ كالمتغير $1 و $2 ولعمل مجموعات يتم إستخدام الرمز الأقواس التالية () أو العلامة التالية | كما يلي

مثال

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

سوف يتم وضع نتائج البحث علي هذا النمط في مجموعات أحدهما للنمط [0-9] والأخري للنمط web

مثال

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

سوف يتم وضع نتائج البحث علي هذا النمط في مجموعات أحدهما للنمط cat والأخري للنمط web ، وهذا النمط يمكن أن يعبر عن "cat" أو "web"

إستخدام كائن النصوص مع الأنماط

بعدما تعلمنا طريقة كتابة الأنماط ، حان الوقت لمعرفة طريقة تطبيقها
لذلك سوف نعتمد علي بعض دوال الكائن النصي لتطبيق الأنماط وهي كما يلي

الدالة match

تستخدم للبحث عن نمط معين بداخل النص ، ثم تقوم بإرجاع مصفوفة بها نواتج البحث

القيمة الراجعة : تقوم بإرجاع مصفوفة

الصيغة العامة

```
String.match( النمط );
```

مثال

```
var str = new String();
str = "hello mr hello";

var arrResult = new Array();

var pattern = /hello/;
arrResult = str.match( pattern );

alert( "عدد مرات التواجد : " + arrResult.length );
```

ويكون الناتج كما يلي

مع أنا النص يحتوي علي كلمة hello مرتين لكن ناتج البحث كان مرة واحدة ، وهذا صحيح لأن النمط المستخدم يتوقف بعد حدوث تماثل له مع النص الذي يتم البحث فيه ، لذلك لإجراء البحث علي جميع النص يتم إستبدال النمط السابق بالنمط التالي كما يلي

مثال

```
var str = new String();
str = "hello mr hello";

var arrResult = new Array();

var pattern = /hello/g;
arrResult = str.match( pattern );

alert( "عدد مرات التواجد : " + arrResult.length );
```

ويكون الناتج كما يلي

تمرين

نريد التأكد من صحة البريد الإليكتروني الذي يتم إستقباله من المستخدم

```
<HTML dir=rtl>
  <Title> البريد الإليكتروني </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    var EmailstrFromUser;
    EmailstrFromUser = prompt("ادخل بريدك الأليكتروني","");

    var emailPattern = /^[a-zA-Z]{1}\w+@[a-zA-Z]{1}\w+\.\w{2,3}/;
    var arrResult = EmailstrFromUser.match(emailPattern);

    if( arrResult != null )
        alert("هذا البريد الإليكتروني صحيح");
    else
        alert("هذا البريد الإليكتروني غير صحيح");
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

تمرين

نريد إستخلاص اسم المستخدم واسم مزود خدمة البريد الإليكتروني لقيمة بريد إليكتروني يتم إستقباله من المستخدم

مع العلم أن اسم المستخدم يسبق موقع الرمز @ و اسم مزود الخدمة يلي الرمز @ فعلي سبيل المثال
إذا كان البريد الإليكتروني المدخل هو a_elhussein@hotmail.com
يكون اسم المستخدم a_elhussein
ويكون اسم المزود hotmail

لاحظ أن النمط المستخدم هو نفس النمط السابق مع إضافة المجموعات بإستخدام الأقواس .

```
<HTML dir=rtl>
  <Title> البريد الإليكتروني </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    var EmailstrFromUser;
    EmailstrFromUser = prompt("ادخل بريدك الأليكتروني","");

    var emailPattern = /^([a-zA-Z]{1}\w+)@([a-zA-Z]{1}\w+)\.(\w{2,3})/;
    var arrResult = EmailstrFromUser.match(emailPattern);
    if( arrResult != null ){
        alert( " : اسم المستخدم" + arrResult[1] + "\r\n" +
               " : اسم المزود" + arrResult[2] );

    }
    else
        alert("هذا البريد اللإليكتروني غير صحيح");
    //-->
    </SCRIPT>
  </HEAD>
</HTML>
```

الدالة replace

تستخدم للبحث عن نمط معين بداخل النص ثم تقوم بإستبداله بنص أخر

القيمة الراجعة : تقوم بإرجاع النص الجديد

الصيغة العامة

```
    String.replace ( نمط البحث , نص الإستبدال );
```

مثال

```
var str = new String();
str = "hello mr hello";

var pattern = /hello/g;
var newStr = str.replace( pattern, "me" );

alert( " : النص الجديد : " + newStr );
```

ويكون الناتج كما يلي

الدالة search

تستخدم للبحث عن نمط معين بداخل النص ثم تقوم بإرجاع مكان تواجدة مثل الدالة indexOf التابعة للكائن النصي

القيمة الراجعة : تقوم بإرجاع موقع النص علي هيئة قيمة عددية

الصيغة العامة

```
String.search( نمط البحث );
```

مثال

```
var str = new String();
str = "hello mr hello";

var pattern = /mr/;
var index = str.search( pattern );

alert( " موقع جملة mr : " + index );
```

ويكون الناتج كما يلي

كائن التعبيرات المنتظمة RegExp

إنشاء كائن التعبيرات المنتظمة RegExp

يمكننا إنشائه كما يلي

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

خصائص الكائن RegExp

الخاصية constructor

سوف نتحدث عن الخاصية constructor بالجزء الثالث من هذا الكتاب ، عند الحديث عن الكائنات والبرمجة الكائنية

الخاصية global

هذه الخاصية للقرأة فقط read only ، وتعبر هل النمط المستخدم تم إستخدام المعامل g به
فعلي سبيل المثال

النمط التالي /hello/g تكون الخاصية global تساوي true
أما النمط التالي /hello/ تكون الخاصية global تساوي false

الخاصية ignoreCase

هذه الخاصية للقرأة فقط read only ، وتعبر هل النمط المستخدم تم إستخدام المعامل i به
فعلي سبيل المثال

النمط التالي /hello/i تكون الخاصية ignoreCase تساوي true
أما النمط التالي /hello/ تكون الخاصية ignoreCase تساوي false

الخاصية multiline

هذه الخاصية للقرأة فقط read only ، وتعبر هل النمط المستخدم سوف لا يتجاهل السطور العديدة به

الخاصية source

هذه الخاصية للقرأة فقط read only ، وتعبر عن صيغة النمط المستخدم بالكائن RegExp .

دوال الكائن RegExp

الدالة compile

تمكننا من إنشاء كائن للتعبيرات المنتظمة به نمط معين

الصيغة العامة

```
RegExp.compile( النمط );
```

أو

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

يحدد المعامل الثاني طريقة البحث بإستخدام النمط
وهنا سوف نشير إلي الرمز m وهي يعبر عن البحث في جميع السطور multiline

مثال

```
var regexpObject = new RegExp();

var regexpObject2 = regexpObject.compile("/me/");

var regexpObject2 = regexpObject.compile("/me/", "i");
```

الدالة test

تستخدم للتأكد من توافق النمط مع النص الممرر لها

القيمة الراجعة : في حالة توافق النمط مع النص ترجع الدالة القيمة true وفي حالة عدم التوافق ترجع false

الصيغة العامة

```
RegExp.test( النص );
```

مثال

```
var RegExp = /me/;
alert( regExp.test("hello me hello") );
```

ويكون الناتج كما يلي

الدالة exec

تستخدم لتنفيذ النمط علي النص الممرر لها

القيمة الراجعة : مصفوفة من كائنات من النوع Match
الصيغة العامة

```
RegExp.exec( النص );
```

مثال

```
 var RegExp = /me/;

var matchArray = RegExp.exec("hello me hello");

if( matchArray == null )
    alert(" لم يتم العثور علي جملة me");
else
    alert( " : تم العثور علي جملة" + matchArray[0] );
```

ويكون الناتج كما يلي

الفصل السادس

التعامل مع التاريخ

سوف نناقش إن شاء الله في هذا الفصل النقاط التالية :

- الكائن Date
- إنشاء كائن التاريخ
- دوال كائن التاريخ

كائن التاريخ Date Object

الجافا سكربت ليس بها نوع بيانات أولي للتاريخ ، لكنك تستطيع أن تستخدم كائن التاريخ ودوالة أو وظائفه للتعامل مع التاريخ والوقت في تطبيقاتك

يتم تخزين التاريخ بالملي سكند milliseconds منذ تاريخ ١ يناير ١٩٧٠

لاحظ أن كائن التاريخ ليس به أي خصائص

إنشاء كائن التاريخ

الصيغة العامة

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

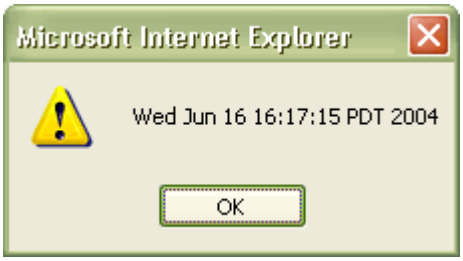
أو

```
var dateObject = new Date( تاريخ );
```

مثال

```
var dateObject = new Date();

alert( dateObject );
```

ويكون الناتج كما يلي

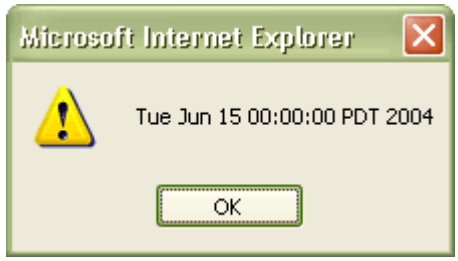


ويمكننا تحديد التاريخ كما يلي

مثال

```
var dateObject = new Date("6/15/2004");

alert( dateObject );
```

ويكون الناتج كما يلي



دوال كائن التاريخ

تنقسم دوال كائن التاريخ إلي ثلاث أقسام أحدهما دوال لإرجاع القيم من التاريخ والأخري لوضع قيم بالتاريخ والأخري لتحويل التاريخ كما يلي

الدالة getDate

تستخدم لإرجاع الأيام الموجودة بالتاريخ من ١ إلي ٣١

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getDate();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getDate() );
```

ويكون الناتج كما يلي

الدالة getDay

تستخدم لإرجاع رقم يوم الأسبوع الموجود بالتاريخ وتكون ممثلة لأيام الأسبوع من ٠ إلي ٦ ، حيث صفر يعبر عن يوم الأحد

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getDay();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getDay() );
```

ويكون الناتج كما يلي

والرقم ٣ يعبر عن يوم الأربعاء

الدالة getMonth

تستخدم لإرجاع رقم الأشهر من ٠ إلي ١١ ، حيث يعبر الرقم صفر عن شهر يناير

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getMonth();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getMonth() );
```

ويكون الناتج كما يلي

والرقم ٥ يعبر عن شهر يونيو

الدالة getYear

تستخدم لإرجاع السنة

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getYear();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getYear() );
```

ويكون الناتج كما يلي

الدالة getHours

تستخدم لإرجاع قيمة عدد الساعات من ٠ إلي ٢٣

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getHours();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getHours() );
```

ويكون الناتج كما يلي

والساعة ١٦ تعادل الساعة الراعة صباحا

الدالة getMinutes

تستخدم لإرجاع قيمة عدد الدقائق من ٠ إلي ٥٩

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getMinutes();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getMinutes() );
```

الدالة getSeconds

تستخدم لإرجاع قيمة عدد الدقائق من ٠ إلي ٥٩

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getSeconds();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getSeconds() );
```

الدالة getTime

تستخدم بحساب التاريخ بالملي سكند الذي مر علي هذا التاريخ منذ تاريخ منتصف ليل يوم ١ يناير ١٩٧٠

القيمة الراجعة : تقوم بإرجاع قيمة عددية

الصيغة العامة

```
Date.getTime();
```

مثال

```
var dateObject = new Date();

alert( dateObject.getTime() );
```

ويكون الناتج كما يلي

الدالة setDate

تستخدم لوضع عدد الأيام بالتاريخ وتكون القيمة المضافة من ١ إلي ٣١

الصيغة العامة

```
Date.setDate( القيمة );
```

مثال

```
var dateObject = new Date();
dateObject.setDate("2");
```

الدالة setMonth

تستخدم لوضععدد الشهور بالتاريخ وتكون القيمة المضافة من ٠ إلي ١١

الصيغة العامة

```
Date.setMonth( القيمة );
```

مثال

```
var dateObject = new Date();
dateObject.setMonth("2");
```

الدالة setYear

تستخدم لتحديد قيمة السنوات بالتاريخ

الصيغة العامة

```
Date.setYear( القيمة );
```

مثال

```
var dateObject = new Date();
dateObject.setYear("2006");
```

الدالة setHours

تستخدم لتحديد قيمة الساعات بالتاريخ

الصيغة العامة

```
Date.setHours( القيمة );
```

مثال

```
var dateObject = new Date();
dateObject.setHours("2");
```

الدالة setMinutes

تستخدم لتحديد قيمة الدقائق بالتاريخ

الصيغة العامة

```
Date.setMinutes( القيمة );
```

مثال

```
var dateObject = new Date();
dateObject.setMinutes("20");
```

الدالة setSeconds

تستخدم لتحديد قيمة الثواني بالتاريخ

الصيغة العامة

```
Date.setSeconds( القيمة );
```

مثال

```
var dateObject = new Date();
dateObject.setSeconds("2");
```

الدالة toGMTString

تستخدم لتحويل التاريخ إلي ما يعادلة بتوقيت جرينتش

الصيغة العامة

```
Date.toGMTString();
```

مثال

```
var dateObject = new Date();

alert( dateObject.toGMTString() );
```

ويكون الناتج كما يلي

الدالة toLocaleString

تستخدم لتحويل التاريخ إلي ما نص علي حسب نظام التشغيل الذي يعمل به الجهاز

الصيغة العامة

```
Date.toLocaleString();
```

مثال

```
var dateObject = new Date();

alert( dateObject.toLocaleString() );
```

ويكون الناتج كما يلي

الدالة parse

تستخدم لتحويل القيمة النصية إلي متغير من نوع تاريخ

الصيغة العامة

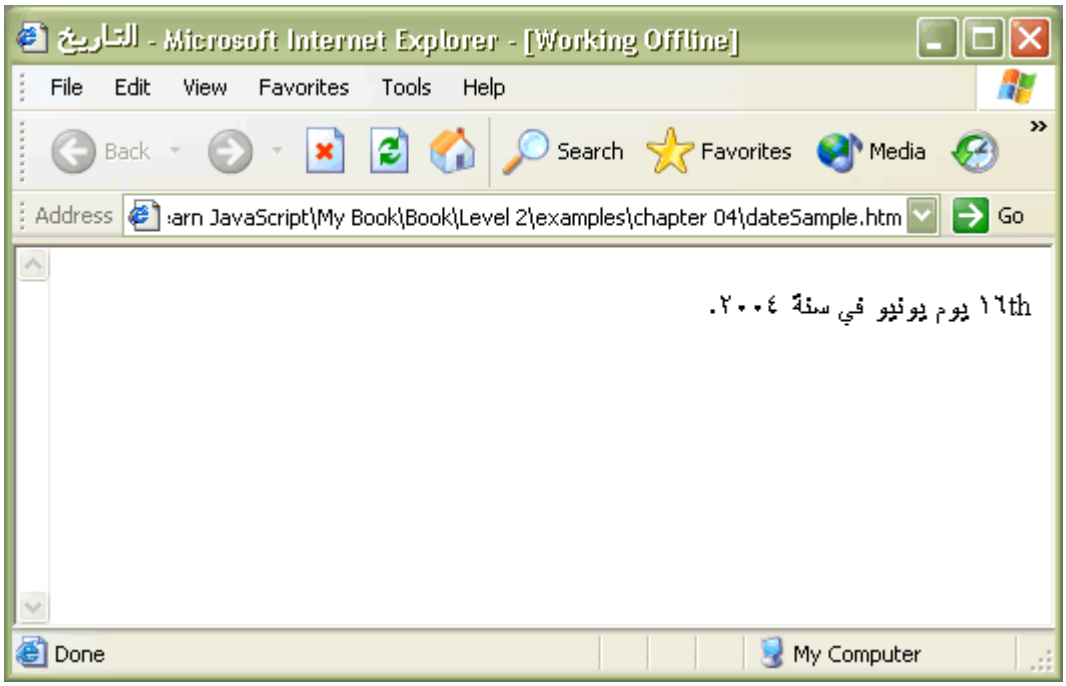
```
Date.parse( "نص التاريخ" );
```

مثال

```
var dateObject = new Date( Date.parse("Wed Jun 16 17:31:01 PDT 2004") );

alert( dateObject.getYear() );
```

ويكون الناتج كما يلي

تمرين

```
<HTML dir=rtl>
  <Title> التاريخ </Title>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript">
    <!--

    var today = new Date();

    var monthName = new Array(11);
    monthName[0] = "يناير";
    monthName[1] = "فبراير";
    monthName[2] = "مارس";
    monthName[3] = "ابريل";
    monthName[4] = "مايو";
    monthName[5] = "يونيو";
    monthName[6] = "يوليو";
    monthName[7] = "اغسطس";
```

```
   monthName[8] = "سبتمبر";
   monthName[9] = "اكتوبر";
   monthName[10] = "نوفمبر";
   monthName[11] = "ديسمبر";

   var myYear = today.getYear();
   if( myYear < 2000 )
       myYear += 1900;


   var myDate = today.getDate();
   var dayExt = "th";

   if( myDate == 1 || myDate == 21 || myDate == 31)
      dayExt = "st";
   else if( myDate == 2 || myDate == 22)
      dayExt = "nd";
   else if( myDate == 3 || myDate == 23 )
      dayExt = "rd";

   var extDate = myDate + dayExt;
   document.write( extDate + " يوم " );
   document.write( monthName[today.getMonth()] + " في سنة " );
   document.write( myYear + "." );
   //-->
   </SCRIPT>
 </HEAD></HTML>
```