بسم الله الرحمن الرّحيم

المحتويات:

1- المقدمة :

في البداية يجب أن تعلم عزيزي القارئ بأني مبتدئ في عالم البرمجة ولست أستاذاً أو خبيراً في هذه اللغة ، و لذلك ورود الأخطاء محتمل ، فإذا وجدت خطأ أرجو منك إبلاغي وتنبيهي عليه في الموضوع الخاص بالشرح في مدونة المادة ، فكل ماهو موجود في هذا الكتيب الصغير هو مجرد اجتهاد ، إن أصبت فمن اللّه وحده ، وإن أخطأت فمن نفسي والشيطان.

لغة الجافاسكربت تتطلب فهم ومعرفة اللغة الأساسية لصفحات الانترنت وهي HTML ، فإذا لم تكن لديك خلفية عنها ، أنصحك بعدم إكمال قراءة هذا الملف والتوجه إلى شرح لغة الـ HTML والموجود في هذا الرابط:
http://aoum150.wordpress.com/2009/07/05/html_lesson

البرمجة بمختلف لغاتها تعتمد على التطبيق والتجربة والكتابة بكثرة ، فإذا لم تكن تنوي أن تستكشف وتجرب وتكتب كل حرف بنفسك أنصحك بعدم إضاعة وقتك في تعلم البرمجة ، لأنك لن تتقنها إلا إذا مارستها ، فالقراءة وحدها لن تكفيك. لذلك أنصح بعدم نسخ أي كود موجود في هذا الكتيِّب ، وإنما قم بكتابته بنفسك حرفاً حرفاً حتى تخطئ وتتعلم من خطئك.

لغة جافاسكربت هي لغة برمجة يتم استعمالها لعمل برامج في صفحات الانترنت، أي انها تعمل في صفحة HTML العادية. وفائدتها هي أنها تسمح لنا بإضافة خصائص إضافية على صفحات الانترنت ، مثل إظهار رسالة ترحيب للمستخدم ، أو أخذ كلمة سر أو بيانات المستخدم ، أو تنبيه المستخدم إذا قام بترك حقل فارغ أو أدخل بيانات خاطئة ، يعني باختصار هي تجعل صفحة الانترنت تفاعلية ، بمعنى أنها تصبح تتجاوب مع أفعال وتصرفات المستخدم المختلفة.

لغة جافاسكربت تختلف كلياً عن لغة الجافا ولا توجد علاقة بينهم ، ولكن أحياناً يتم استخدام كلا اللغتين في بعض تطبيقات وبرامج الانترنت.

كل لغة برمجة لها قوانين وقواعد لكتابتها تسمى syntax ، إذا لم نتّبع هذه القوانين بشكل صحيح وصارم فإن احتمال عدم عمل البرنامج ، أو عمله بغير الشكل الذي نريده ونتوقعه أمر وارد بشكل كبير ، لذلك معرفة قوانين وقواعد كتابة اللغة من أهم الأساسيات التي يتعلّمها المبرمج في لغات البرمجة.

لغة الجافاسكربت – وإن لم يخب ظني جميع لغات البرمجة المشهورة – تكون حسّاسة بالنسبة للأحرف الكبيرة والصغيرة ، بمعنى أنها ستعامل الكلمات if – IF – If - iF كأنها أربع كلمات مختلفة ، فيجب التنبه لذلك.

الأخطاء في أي عمل ، وفي البرمجة بشكل خاص واردة بشكل كبير ، فمن لم يخطئ ومن يخاف من الخطأ لن يتعلّم ، فلا يوجد مبرمج لم يتعرّض لأخطاء أثناء البرمجة ، بل إن من أجزاء تعليم البرمجة هو تعليم تصحيح الأخطاء واكتشافها. ولكن في نفس الوقت من لا يستفيد ويتعلّم من أخطائه لن ينجح ، فالمبرمج الناجح هو الذي يتعلّم من أخطائه ويتجاوزها بسرعة ، وكثرة التجارب والتطبيق هي المفتاح لهذا الشيء. و الأخطاء في عالم البرمجة تسمى bugs ، وعملية تصحيحها واكتشافها تسمى

.debugging

الجافاسكربت ممكن أن تكون خطيرة ومؤذية لأجهزة الحاسب والمتصفِّحات إذا تمّ استخدامها بشكل سيء ، فمن الممكن على سبيل المثال أن نكتب أوامر تنفيذية على الكمبيوتر الزائر للصفحة تقوم بتخريب أو تعطيل الجهاز أو أحد محتوياته.

في جميع لغات البرمجة ، يتم تنفيذ الأوامر بالترتيب ، بمعنى أن الكمبيوتر سيبدأ بتنفيذ الأمر الموجود في السطر الأول ، ثم السطر الثاني ... وهكذا حتى ينتهي من تنفيذ الأمر الموجود في آخر سطر في البرنامج ، ثم سينتهي.

2- من أساسيات البرمجة: تقييم المعادلات والمقارنات

تقييم المعادلات أو المقارنات هو من مهام الكمبيوتر والبرامج الرئيسية ، فمن خلالها يتم عمل البرامج الصغيرة والكبيرة ، على سبيل المثال بعض المواقع التي لاتسمح بزيارة الأشخاص الذين تقل أعمارهم عن 18 سنة تستعمل هذه المقارنة ، فنجدها تطلب من الزوار كتابة تاريخ ميلادهم مثلاً ، ثم تعمل مقارنة بين العمر المدخل ورقم 18 ، فإذا كان العمر المدخل أكبر من 18 ، سيتم نقلهم إلى صفحة الموقع الرئيسية وإذا كان العمر أقل سيتم نقلهم إلى صفحة أخرى تحتوي على رسالة إعتذار على سبيل المثال.
هذه العملية تتم بشكل آلي عن طريق البرنامج ، فبمجرد أن أدخلنا إليه بعض البيانات أصبح يستطيع أن يقارن و يقيِّم ثم ينفِّذ النتيجة المناسبة لكل معادلة أو مقارنة.

في الكمبيوتر ، كل حرف أو رمز يكون له قيمة رقمية خاصة تابعة للترميز العالمي ASCII code أو Unicode . فعندما نعمل مقارنات بين أحرف أو رموز أو بين أحرف وأرقام مثلاً ، فإن الكمبيوتر في الواقع يعمل مقارنة بين قيمة هذه الأحرف أو الرموز في ترميز ASCII وليس بالحرف أو الرمز نفسه.
الفرق بين ترميز آسكي ويونيكود هو أن الثاني يحتوي على حروف وأشكال للغات أكثر بكثير من الأول ، فاليونيكود يحتوي على 65536 مابين أرقام وحروف ورموز مختلفة لعشرات اللغات ومن ضمنها العربية ، بينما يحتوي ترميز آسكي على 128 فقط! مابين أحرف وأرقام ورموز.
قيمة أي حرف أو رمز في الترميزين هي نفسها ، لذلك لا داعي للتساؤل عن قيمة حرفٍ أو رمزٍ ما في كلا النوعين.
القيم في ترميز آسكي متسلسلة بشكل ثابت مع تسلسل الأحرف والأرقام والرموز ، وبشكل عام نتائج التقييم في هذا الترميز ستكون كالتالي:

```
. < 0 < 1 < 2 < ... < 9 < A < B < ... < Z < a < b < ... < z
```

آلية تقييم العمليات الحسابية في الكمبيوتر:

العمليات الحسابية في أجهزة الحاسب يتم تنفيذها حسب قوانين معينة تسمى أولويات التنفيذ على النحو التالي:
أولاً: تنفيذ مابين الأقواس.
ثانياً: تنفيذ عمليات الضرب والقسمة.
ثالثاً: تنفيد عمليات الجمع والطرح.

- في حالة تشابه العمليات سيتم التنفيذ ابتداءً من اليسار ثم سيتجه لليمين حتى ينتهي.
- في بعض لغات البرمجة (الجافاسكربت ليست من ضمنها) سيتم تقييم المعادلات من اليسار لليمين بغض النظر عن نوع العمليات الموجودة.

مثال:

```
2 - 5 +3 * 6 / (2 +1)
```

كم تعتقد ستكون نتيجة هذه المعادلة من وجهة نظر الكمبيوتر؟

سنأخذها بالأولويات السابقة ونحلل المعادلة خطوة خطوة ..

أولاً : سنبحث عن الأقواس إن وجدت ونقوم بتنفيذها:

```
2 – 5 + 3 * 6 / 3
```

ثانياً : سنبحث عن عمليات الضرب والقسمة وننفذها ابتداءً من اليسار ثم ننتقل لليمين حتى ننتهي:

```
2 – 5 + 18 / 3
2 – 5 + 6
```

ثالثاً : سنبحث عن عمليات الجمع والطرح وننفذها من اليسار لليمين:

```
-3 + 6
3
```

طيب بما أننا عرفنا الآن آلية تقييم المعادلات سننتقل لمرحلة المقارنات ..

نتائج المقارنات دائماً إما أن تكون المقارنة صحيحة أو خاطئة لايوجد خيار ثالث، سأقسم المقارنات إلى ثلاثة أقسام:

1- مقارنة أرقام بأرقام:

مقارنة الأرقام هي الأسهل، وهي تعتمد على أولويات تنفيذ العمليات الحسابية التي شرحتها قبل قليل.

مثال: قيم العبارة التالية:

```
15 – (3 + 3) * 2 < 3 * 3 + 1
```

سنحلل كل طرف ثم نقيّم الناتج النهائي للطرفين هل هو صحيح أم خاطئ:

```
15 – (3 + 3) * 2 < 3 * 3 + 1
15 – 6 * 2 < 9 + 1
15 – 12 < 10
3 < 10
```

العبارة صحيحة لأن 3 أصغر من 10 !

2- مقارنة أحرف بأحرف:

كما ذكرت سابقاً ، الكمبيوتر يتعامل مع الأحرف والرموز بقيمتها التابعة لترميز ASCII ، لذلك عندما نعمل مقارنة أرقام مع أحرف سيتم مقارنة قيمهم التابعة لهذا الترميز ، وكما ذكرت في المقدِّمة ، قاعدة هذا الترميز هي كالتالي:

```
. < 0 < 1 < 2 < ... < 9 < A < B < ... < Z < a < b < ... < z
```

وعلى أساسها بإمكاننا عمل أي مقارنة ، سواء كانت بين أحرف أو بين أحرف وأرقام أو بين أرقام ورموز وغيره ، ولكن يجب التنبه بأن الأرقام التي أتحدث عنها يجب أن توضع بين العلامتين '' حتى تنطبق عليها القاعدة التي في الأعلى.
وفي مقارنة الكلمات سيتم عمل المقارنة على أول حرف في الطرفين، وإن كانا متشابهين سيتم مقارنة الحرفين التاليين لهما وهكذا حتى يستطيع الحصول على نتيجة.

مثال: قيم العبارة التالية:

```
'elephant' < 'mouse'
```

المقارنة ستتم بين الحرفين e و m ..
وفي ترميز آسكي ، قيمة e أقل من قيمة m ، لذلك العبارة صحيحة !

طيب لو غيرنا أول حرف من إحدى الكلمات وجعلناه كبيراً:

```
'elephant' < 'Mouse'
```

المقارنة ستكون بين e و M ..
النتيجة هنا خاطئة ، لأن حرف M الكبير قيمته أكبر من قيمة حرف e الصغير.

لو كان أول حرفين في الطرفين متشابهين ، ستتم المقارنة بين الحرفين الذين بعدهم ، مثال:

```
'cat' < 'cave'
```

هنا الحرفين الأولين في الكلمتين متشابهين ، لذلك ستتم المقارنة بين t و v ..
والعبارة صحيحة لأن t أصغر من v !

أحب التنويه هنا بأن العلامة ' التي تحيط بالكلمات ، تدلّ على أن المقارنة بين `string` و `string` آخر ، وسيتم توضيحها لاحقاً في قسم المتغيّرات Variables.

3- مقارنة أرقام بأحرف أو العكس:

تعتمد على نفس المبدأ والقاعدة، مثال:

```
'7' < 'a'
```

النتيجة صحيحة ، لأن قيمة الرقم 7 أقل من قيمة الحرف a.

مثال آخر:

```
7 == '7'
```

أحب أذكركم هنا بأن الكمبيوتر يتعامل مع قيم الأرقام والأحرف وليس بشكلها ، والآن في هذا المثال يوجد رقم 7 في طرف وفي الطرف الثاني 7 على شكل `string` لأنها بين قوسين.
وفي كلا الحالتين ، سيقوم الكمبيوتر بتحويل الطرفين إلى قيمتهم الفعلية ثم سيقارن بين القيم ، لذلك المقارنة صحيحة هنا.

نقطة أخيرة بخصوص مقارنة الأرقام والأحرف، المقارنات التالية:

```
7 < 'a'
7 > 'a'
7 == 'a'
```

كلها خاطئة !

بصراحة لا أعلم مالسبب بالضبط ، ولكني جربتها شخصياً وظهرت لي نتائجها كلها خطأ ، أعتقد والله أعلم بأن السبب هو أنه لا يمكننا مقارنة رقم بـ string مع أننا قمنا بذلك في المثال الأخير ، ولكن يبدو أنه كان حالةً استثنائية لأن الطرفين يحتوون على أرقام ، لأننا لو جربنا المقارنة التالية 7 < '9' سنجد النتيجة صحيحة . هذه مجرد توقعاتي الشخصية وهي قابلة للخطأ أكثر من الصواب ، لكن أحببت أن أوضح وجهة نظري في هذا الموضوع ، وإن شاء الله لن نحتاج في منهجنا لمثل هذه التعقيدات الزائدة ، لكن تم ذكر هذه النقطة للتوضيح.

أحب أن أذكر هنا بأن علامة أكبر من أو يساوي في لغات البرمجة تكتب علامة يساوي في اليمين وعلى يسارها أكبر من متصلين ببعض، مثلاً لو أردنا كتابة 3 أكبر من أو يساوي 2 نكتبها على الشكل التالي في البرنامج:

```
3 >= 2
```

ولو أردنا كتابة 5 أكبر من أو تساوي 3 :

```
5 >= 3
```

واذا أردنا كتابة 5 تساوي 5 نكتبها على الشكل التالي:

```
5 == 5
```

علامتين يساوي وليست واحدة.

وإذا أردنا كتابة 5 لاتساوي 4 نكتبها على الشكل التالي:

```
5 != 4
```

علامة التعجب تفيد النفي ، أي أنه لايساوي.

وللتلخيص هذا جدول لإشارات المقارنة مع معانيها:

النتيجة	مثال	معناها	إشارة المقارنة
خطأ	5 < 6	أصغر من	<
صح	2 <= 2	أصغر من أو يساوي	<=
صح	1 > 3	أكبر من	>
خطأ	3 >= 4	أكبر من أو يساوي	>=
صح	1 == 1	يساوي	==
خطأ	1 != 1	لا يساوي	!=

وفي الجدول التالي توضيح لإشارات العمليات الحسابية بافتراض أن y = 5 و x = 10 :

الإشارة	معناها	مثال 1	مثال 2
*	ضرب	x * y = 50	y * x = 50
/	قسمة	x / y = 2	y / x = 0.5
+	جمع	x + y = 15	y + x = 15
-	طرح	x – y = 5	y – x = -5
%	باقي القسمة	x % y = 0	y % x = 5

بالنسبة لإشارة باقي القسمة % فهي غير مشروحة في المنهج ولكنها مهمة جداً في عالم البرمجة، وللأسف جاءت لنا في الاختبار النهائي ، ولذلك سأشرحها احتياطاً.

سأستخدم الأمثلة مباشرةً لتوضيحها وشرحها ..

21 % 7 = ?

هي تعتمد على القسمة أولاً ..

يعني نقسم 21 على 7 والنتيجة رقم 3

لذلك لايوجد باقي للقسمة فتكون نتيجتها 0

21%7 = 0

20 % 3 = ?

هنا نقسم 20 على 3 والنتيجة 6.67

لكن هنا لانريد عدداً غير صحيح ، لذلك سنأخذ الرقم 6 فقط

3 ضرب 6 تساوي 18

معناها باقي قسمة 20 على 3 يساوي 2

20%3 = 2

10%3 = ?

نفس الشي نقسم 10 على 3 .. النتيجة ستكون 3.33

نأخذ 3 بدون الكسر ونضربها في 3 ، وبعدها نحسب الفرق بين النتيجة وبين رقم 10 وستكون هي باقي القسمة

يعني باقي القسمة ستكون: (3*3) - 10

10%3 = 1

لو كانت قسمة الرقمين على بعض عدد صحيح ستكون نتيجة باقي القسمة صفر..

20%5 = ?

نقسم 20 على 5 ، والنتيجة ستكون 4 بدون كسور وأرقام عشرية. لذلك لن يكون هناك باقي للقسمة.

```
20%5 = 0
```

مثال أخير:

```
12%7 = ?
```

نقسم 12 على 7 والنتيجة هي 1.71 تقريباً .. نأخذ 1 ونترك الكسر ، ثم نضربه في 7 ، ونحسب الفرق بين النتيجة وبين رقم 12 . الفرق هو 5 .. لذلك :

```
12 % 7 = 5
```

3- من أساسيات البرمجة : or , and , not :

هذه العبارات تستخدم لربط عدة مقارنات مع بعض وهي مفيدة جداً وأعتقد بأنه لايوجد برنامج يخلو منها.

1- or :

تكتب or في البرمجة على شكل || ، والزر الخاص بهذا الرمز موجود في لوحة المفاتيح على يسار زر الإدخال (إنتر) ويجب الضغط في نفس الوقت على زر shift.

هذه الأداة وظيفتها تخيير المتصفح بين مقارنتين أو أكثر، فلو كانت إحداهما صحيحة ستكون النتيجة النهائية صحيحة، أي أنه لن تكون النتيجة النهائية خاطئة - إذا كانت جميع العلاقات مربوطة بهذه الأداة - إلا اذا كانت كل المقارنات نتيجتها خاطئة.

في الجدول التالي توضيح أكثر ، بافتراض وجود مقارنتين ، وسنرمز لهم بـ A , B :

A	B	A || B
True	True	True
True	False	True
False	True	True
False	False	False

معنا السطر الأول كالتالي: إذا كانت المقارنة الأولى صحيحة أو المقارنة الثانية صحيحة فالنتيجة النهائية صحيحة.
معنا السطر الثاني كالتالي: إذا كانت المقارنة الأولى صحيحة أو المقارنة الثانية خاطئة فالنتيجة النهائية صحيحة.
وهكذا لبقية الأسطر ..

2- and :

تكتب and على شكل && ، وهي تربط بين أكثر من معادلة أو مقارنة ، ونتيجتها لا تكون صحيحة ، إلاّ إذا كانت جميع المقارنات – التي تربط بينها – صحيحة. وللتوضيح أكثر انظر للجدول التالي:

A	B	A && B
True	True	True
True	False	False
False	True	False
False	False	False

معنى السطر الأول: إذا كانت المقارنة الأولى صحيحة و المقارنة الثانية صحيحة فالنتيجة النهائية صحيحة.

معنى السطر الثاني: إذا كانت المقارنة الأولى صحيحة و والمقارنة الثانية خاطئة فالنتيجة النهائية خاطئة.

الخ ...

3- not :

تكتب في البرمجة على شكل علامة تعجب ! وتفيد بعكس نتيجة المقارنة ، أي أنه إذا كانت نتيجة المقارنة صحيحة فالأمر هذا يعكس النتيجة إلى خاطئة ، كما في الجدول:

A	!(A)
True	False
False	True

في الجدول التالي ، تطبيق لجميع الأوامر: && and , ! not , || or :

A	B	A || B	A && B	! (A)	! (A || B)	! (A && B)
True	True	True	True	False	False	False
True	False	True	False	False	False	True
False	True	True	False	True	False	True
False	False	False	False	True	True	True

مثال:

سؤال أتى لنا في الاختبار النهائي ، يقول أثبت عن طريق جداول Truth tables أن:

```
NOT (A AND B) = NOT A OR NOT B
```

أو ممكن تكتب بالشكل التالي:

```
! (A && B) = !A || !B
```

طريقة الحل بسيطة ان شاء الله اذا كنا فاهمين طريقة عمل كل أمر.

أول شيء ، لحل مثل هذه الأسئلة ، سنقوم بعمل عمودين ثابتين مهما كانت المعادلات المطلوب إثباتها ، كل عمود خاص بمقارنة ، وسنضع تحتها جميع الاحتمالات لها:

A	B
True	True
True	False
False	True
False	False

الذي عملته هنا ثابت لحل أي مسألة من هذا النوع ، حتَّى نشمل جميع الاحتمالات للمعادلات.

طيب الآن نرجع للمعادلة التي في السؤال ، ونحلل كل جزء فيها ..

الجزء الأيسر: حتّى نصل له لابد من إيجاد أجزائه الصغيرة أولاً ..

```
! (A && B)
```

أول شيء لازم نحسب نتائج A&&B حتّى نستطيع حساب نتائج ! (A&&B) ، لذلك سنضيف عمودين على الجدول ، كل عمود خاص بواحدة ، وسنضع النتائج في صفوفها كما تعلّمنا:

A	B	A && B	! (A && B)
True	True	True	False
True	False	False	True
False	True	False	True
False	False	False	True

كذا نكون انتهينا من الجزء الأيسر، فننتقل للجزء الأيمن ونحلله بنفس الطريقة ..

```
!A || !B
```

أول شيء ، لازم نحسب نتائج !A و !B حتّى نستطيع حساب نتائج !A || !B ..

فالآن سنقوم بإضافة ثلاثة أعمدة للجدول ، عمود لـ !A و عمود لـ !B وعمود لـ !A || !B :

A	B	A && B	! (A && B)	!A	!B	!A || !B
True	True	True	False	False	False	False
True	False	False	True	False	True	True
False	True	False	True	True	False	True
False	False	False	True	True	True	True

وبهذا نكون أثبتنا أن الطرفين يتساوون في جميع الحالات ، وهذا الجدول هو الحل للسؤال.

قاعدة:

دائماً يتم تنفيذ ! قبل && أو || ، مثل الجزء الأيمن في المثال !A || !B . لكن إذا كانت && أو || داخل أقواس مثل الجزء الأيسر في المثال ! (A && B) ، نقوم بتنفيذ مابداخل القوس أولاً.

4- المتغيرات Variables :

المتغيِّر هو معرِّف له اسم وله نوع معينّ (يحددهما المبرمج) ، هذا المتغيِّر يرتبط بقيمة معينة مخزنة في ذاكرة الحاسب ، هذه المتغيرات يمكن أن تكون قيمتها الموجودة في الذاكرة عبارة عن أرقام أو كلمات أو جُمل ، فمن الممكن أن تكون مثلاً تكاليف مالية ، أو أسماء طلاّب ، أو درجات طلاّب ، أو أرقام هواتف ، أو عناوين ، أو أي شيء آخر ممكن أن يفيدنا في تحقيق الهدف من البرنامج. وسميَّت متغيرات ، لأننا نستطيع تغيير قيمها في أثناء عمل البرنامج.

في الجافا سكربت ، كلُّ المتغيرات باختلاف أنواعها المشروحة في المنهج تكون من نوع واحد – بعكس باقي لغات البرمجة – ولكن يوجد شيء بسيط يجب الانتباه له ، وهو إذا أردنا أن نتعامل مع جُمل أو كلمات أو أرقام سرية أو أي شيء لا يصلح أو لانريد استعماله في عمليات الحساب الرياضية (مثل الجمع ، الضرب .. الخ) يجب علينا تمييزه عن الأرقام عن طريق وضعه بين علامات 'التنصيص' عند تعريفه ، وهو يسمى في البرمجة `String`.

فالمتغير له اسم وقيمة ، فمثلاً من الممكن أن يكون لدينا متغيِّر اسمه `userName` وقيمته هي `'Learner'` ، أو يكون اسمه `discount` وتكون قيمته `0.20` ، أو يكون اسمه `studentGrade` وتكون قيمته `95` ، لاحظوا بأن القيمة الأولى وضعتها بين علامتي `'تنصيص'` لأني أريد التعامل معها على شكل `string` ، أو بمعنى آخر لا أريد أن أعدِّل على هذه القيمة أو أستعملها في المعادلات الرياضية ، ولكن القيمتين الثانية والثالثة جعلتها وحيدة بدون علامات 'التنصيص' لأني أريد أن أستخدم هذه الأرقام في معادلات رياضية مختلفة ، فمثلاً أريد من البرنامج أن يقوم بمقارنة درجة الطالب حتى يقوم بوضع معدَّل مكانها حسب القوانين التي وضعتها في برنامج ، وبالنسبة لسعر الخصم ، أريد أن يتم ضربه في الأسعار الأصلية ومن ثم يعرض لي السعر بعد الخصم.

المتغيرات يتم عملها في مرحلتين ، مرحلة الإعلان **declare** ، ومرحلة الإنشاء أو البدء **initialize**.
1- المرحلة الأولى declare : نقوم بإخبار الكمبيوتر عن طريق كتابة أمر معيَّن بأننا قمنا بعمل متغيِّر ونريده أن يحجز مكان له في ذاكرة الحاسب. في لغات البرمجة الأخرى تعتمد المساحة التي حجزها الكمبيوتر لهذا المتغيِّر على نوع المتغيِّر ، فكما قلت في اللغات الأخرى توجد عدّة أنواع للمتغيرات.
2- المرحلة الثانية initialize : سنقوم هنا بوضع قيمة للمتغيِّر الذي عملناه في المرحلة الأولى ، وسيقوم البرنامج بوضع هذه القيمة في مكان الذاكرة الذي تم حجزه في المرحلة الأولى.

تعريف المتغيرات أو الإعلان عنها في الجافاسكربت يكون ببساطة عن طريق كتابة `var` ثم اسم المتغيِّر ، فلو أردنا تعريف متغير باسم `stundentGrade` نكتبه كالتالي:

```
var studentGrade;
```

ولو كنّا نريد تعريف متغيّر من نوع `string` باسم `studentName` سيكون الأمر كالتالي:

```
var studentName = new string();
```

نلاحظ بأن الإعلان عن متغيِّر من نوع string يتكون من جزأين ، الجزء الذي بعد علامة اليساوي يخبر البرنامج بأن هذا المتغير سيكون من نوع string.

وأيضاً نلاحظ وجود علامة فاصلة منقوطة ; بعد نهاية كل أمر ، هذه الفاصلة ضرورية جداً في كل أوامر البرنامج ، لأن البرنامج عن طريقها يتعرف على بداية ونهاية كل أمر برمجي ، إذا لم تكن موجودة من الممكن أن يعمل البرنامج بشكل عادي ، ولكن غالباً لن يعمل ، لأنه قام بوصل أمرين برمجيين مع بعض.

نستطيع أن نقوم بالإعلان عن أكثر من متغيّر في نفس الأمر ، مثل:

```
var studentName, studentGrade, studentSection;
```

المرحلة الثانية وهي إنشاء المتغيِّر ، تكون عن طريق وضع قيمة للمتغيِّر باستخدام علامة يساوي ، مثل:

```
studentGrade = 95;
```

وللمتغيرات من نوع string ستكون كالتالي:

```
studentName = 'Learner';
```

علامة التنصيص كما ذكرت ، تبيِّن لنا وللبرنامج بأن هذا المتغير هو string.

قيم المتغيرات ، ممكن أن تكون قيم لمتغيرات أخرى ، مثلاً لو قمنا بتعريف متغير في بداية البرنامج باسم grade وأعطيناه قيمة 92 ، ومن ثم قمنا بتعريف متغير جديد وسميناه studentGrade ، إذا أردنا أن نجعل قيمة المتغير الثاني تساوي قيمة المتغير الأول ، نكتبها كالتالي:

```
studentGrade = grade;
```

وستكون الآن قيمة المتغير الثاني تساوي 92 ، ونفس الكلام ينطبق على المتغيِّرات من نوع string.

يمكننا اختصار مرحلتي الإعلان والإنشاء – وهذا هو المتعارف عليه لدى المبرمجين – كالتالي:

```
var studentGrade = 95;
var studentName = 'Learner';
```

نلاحظ بأن الفرق الوحيد الذي يميز المتغيرات من نوع string عن باقي المتغيرات هو علامتي 'التنصيص' فقط.

شروط و قوانين تسمية المتغيرات:

توجد قوانين وشروط لكتابة أسماء المتغيرات ، تجاوز هذه الشروط ممكن أن يمنع البرنامج من العمل بشكل صحيح ، ومن الممكن أن لا يؤثر على عمله ، ولكن اتِّباع الشروط هو شيء مفضَّل لمن يريد الاحتراف في البرمجة ، لأنه سيجعلك تتواصل مع المبرمجين الآخرين بشكل أفضل ، وسيجعل برامج أسهل للفهم للمبرمجين الذين سيقرأون الكود.

هذه الشروط خاصَّة بلغة الجافاسكربت ، وهي في الغالب تتشابه مع باقي لغات البرمجة المختلفة.

- الحرف الأول: يجب أن يكون حرف كبير أو صغير أو علامة underscore _ أو رمز الدولار $. ومن المفضل أن لايكون الحرف الأول حرف كبير.
- بقية الأحرف: يجب أن تكون حرف كبير أو صغير أو علامة underscore _ أو رمز الدولار $ بالإضافة إلى إمكانية استعمال الأرقام.
- إذا كان الاسم يتكون من أكثر من كلمة ، لاتستعمل المسافات أبداً ، واجعل جميع أحرف الكلمات صغيرة باستثناء الحرف الأول من الكلمة الثانية والثالثة والرابعة .. الخ ، مثل: `studentGrade` ، `carModelSerialNumber` ، `schoolStudentGrade`

أو يمكنك فصل الكلمات عن بعض بعلامة underscore _ إذا أردت.

- يجب تسمية المتغيرات بأسماء تدلّ على فائدة المتغيّر ، بمعنى إذا أردت أن تعمل متغير ليحمل اسم مستخدم ، اجعل اسمه userName ، ولا تقم بتسميته u1 مثلاً أو un ، لأنك إذا أردت مراجعة الكود بعد زمن ، ستجد صعوبة في معرفة فائدة هذا المتغير إذا لم تقم بتسميته بشكل صحيح وواضح.
- يجب أن تتأكد من أن الاسم لايعتبر كلمة محجوزة في الجافاسكربت (reserved word). وللاطلاع على الكلمات المحجوزة في الجافاسكربت انظر الرابط التالي:

http://www.javascriptkit.com/jsref/reserved.shtml

5- من أساسيات البرمجة: if..else , while , for :

في البرمجة يوجد نوعين من الأوامر الهامة:

1- أوامر التكرار مثل for, while :

ومهمتها هي أن تقوم بتكرار عملية معينة حتى ينطبق شرط معين (نحدده نحن) ويتوقف الأمر عن التكرار، طبعاً توجد أوامر تكرار أخرى ، لكن في المنهج لم يتم شرحها لذلك سأكتفي بما هو موجود في منهج المادة.

2- أوامر الشرط أو الاختيار مثل if...else :

هذا هو الأمر الوحيد المشروح في المنهج بما يتعلق بأوامر الاختيار وهو كما يتبين من اسمه ، يفيد التخيير ، أي نجعل البرنامج عن طريق هذا الأمر يقوم بالاختيار بين عدة بدائل أو حالات مختلفة.

1- for :

يتكون هذا الأمر من ثلاثة نقاط أساسية يجب أن يحتوي عليها:

1- تعريف المتغير وتعيين قيمة له ، هذا المتغير سنجعله يتحكم بعدد مرات تكرار الأوامر الموجودة في محتوى أمر for.

2- مقارنة أو معادلة ، إذا كانت قيمتها صحيحة سيتم تنفيذ الأوامر الموجودة في محتوى for ، واذا أصبحت خاطئة سيتوقف الأمر عن التكرار. طبعاً هذه المقارنة يجب أن تحتوي على المتغير الذي قمنا بتعريفه حتى نتمكن من التحكم بعدد مرات التكرار.
أعرف بأن الأمر غير واضح الآن ولكن تابعوا معي وسيتضح بإذن الله،

3- كتابة معادلة لتغيير قيمة المتغير الذي عرفناه في الجزء الأول، هذه المعادلة سيتم تطبيقها بعد نهاية كل دورة تكرار وفائدتها هي انها ستزيد مثلاً من قيمة المتغير في كل مرة حتى نستطيع أن نجعل قيمة المقارنة (الموجودة في الجزء الثاني) خاطئة لنوقف عملية التكرار.

لو أردنا عمل برنامج يكتب رمز $ عشر مرات في سطر واحد في المتصفح ستكون طريقة كتابة هذا الأمر كالتالي:

أولاً : سنكتب الجزء الأول وهو إعلان وإنشاء المتغير – الذي سيتحكم بعدد مرات تكرار أمر الكتابة - مع قيمة يبدأ منها العد كالتالي:

```
var count = 1
```

ثانياً : سنكتب معادلة أو مقارنة لإيقاف أمر التكرار بعدما تتم كتابة الرمز عشر مرات:

```
count <= 10
```

لماذا جعلناها أصغر من أو يساوي؟

افترض بأننا وضعناها count = 10 .. هذه المقارنة ستكون خاطئة منذ بداية البرنامج لأن قيمة المتغير في البداية ستكون 1.

افترض بأننا وضعناها count > 10 .. هذه المقارنة ستكون خاطئة منذ بداية البرنامج لأن قيمة المتغير في البداية ستكون 1.
افترض بأننا وضعناها count < 10 .. هذه المقارنة ستكون صحيحة إلى أن تصبح قيمة count تساوي 9 وبعدها ستكون المقارنة خاطئة ، أي أن التكرار سيكون 9 مرات فقط ، ونحن نريده أن يتكرر 10 مرات.

ثالثاً : سنكتب معادلة لزيادة قيمة المتغير بواحد:

```
count = count + 1
```

ويمكننا كتابتها أيضاً هكذا:

```
count += 1 أو count++
```

وطريقة الكتابة الأكثر استعمالاً في عالم البرمجة هي count++ .
هذه المعادلة ستزيد من قيمة المتغير برقم واحد بعد تنفيذ كل عملية (كتابة رمز $) حتى تصبح قيمة المتغير 11 ، وسيتوقف بعدها أمر التكرار لأن الشرط والمقارنة ستصبح خاطئة.

طيب الآن سنكتب أمر لكتابة الرمز $ (هذا الأمر غير صحيح ولكنه للتوضيح فقط) :

```
write $;
```

طيب الآن نجمع الأجزاء السابقة مع بعض:

```
for (var count=1; count<=10; count++)
write $;
```

طريقة تنفيذ هذا الأمر ستكون كالتالي:
سيقوم البرنامج بالذهاب للجزء الأول ثم سينتقل للجزء الثاني وإذا كانت نتيجة المقارنة صحيحة سينتقل إلى الأمر الذي نريد تنفيذه (المحتوى). ثم سينتقل للجزء الثالث ليزيد قيمة المتغيّر count ، ثم سيعود للجزء الثاني ، وهكذا ...

وإذا كانت نتيجة المقارنة خاطئة لن يتم تنفيذ الأمر وسيتم انهاء البرنامج ، أو الانتقال لأمر آخر إذا كان البرنامج يحتوي على أوامر أخرى.
طبعاً الأجزاء الثلاثة التي يتكوّن منها أمر for تفصل بينها فاصلة منقوطة ; ويكونون جميعهم في داخل (قوسين) .
نقطة أخيرة.. الأمر أو الأوامر التي نريد تنفيذها في داخل أمر التكرار يجب أن نضعها بين قوسين { } لأنها تبين للبرنامج أين يبدأ محتوى for أو while أو if وأين ينتهي، يعني للمثال الذي كتبته ، المفروض نكتبه بالشكل التالي:

```
for (var count=1 ; count<=10 ; count++)
{
    write $;
}
```

للتلخيص:

الجزء الأول سيتم تنفيذة أول مايبدأ الأمر ومن ثم لن يعود له البرنامج مرة أخرى.

ثم سيبدأ البرنامج بعمل دورة تبدأ من الجزء الثاني ثم المحتوى ثم الجزء الثالث ..

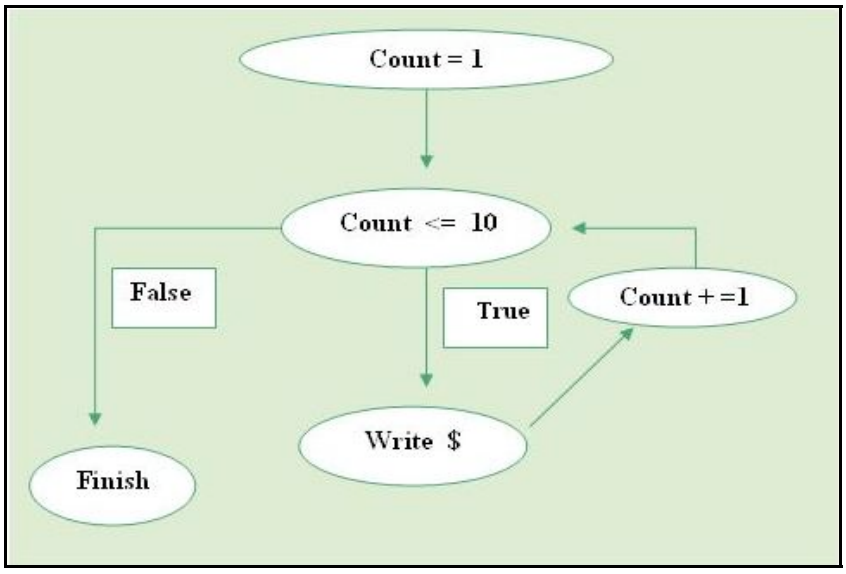
الجزء الثاني >> المحتوى >> الجزء الثالث

الجزء الثاني >> المحتوى >> الجزء الثالث

...

إلى أن يصبح الشرط الموجود في الجزء الثاني غير صحيح ، فيتوقف عندها أمر التكرار.

طريقة عمل الأمر السابق موضحة في الرسم التالي:



إذا كانت طريقة عمل أمر for واضحة ، ستكون بقية الأوامر سهلة إن شاء الله ، لأنها هي الأكثر تعقيداً من بينهم.

2- while :

نفس الشيء ، تستخدم اذا أردنا تكرار عملية أو أمر معين عدة مرات ، ولكنها أقل تعقيداً من أمر for.

وهي تحتوي على جزء واحد يكون عبارة عن مقارنة أو شرط أو اختبار، إذا كان الشرط صحيح سيتم تنفيذ الأوامر الموجودة في محتوى while ، وإذا كان خاطئ لن يتم تنفيذها.

طبعاً يمكننا تنفيذ أمر كتابة رمز $ عشر مرات عن طريق while أيضاً ، فاستعمال for أو while يعتمد على فكر المبرمج غالباً ، ولكن بشكل عام ، أمر for سنستخدمه إذا كنّا سنكرر عملية ما بعدد مرات معروفة مسبقاً ، لكن لو كنّا نرغب مثلاً في أخذ رقم سري من المستخدم وسنقوم بتكرار إظهار نافذة الإدخال حتى يقوم بكتابة الرقم السري بشكل صحيح ، سنستخدم أمر while ، لأننا لا نريد تحديد عملية التكرار بعدد معينّ ، وإنما سنحدده بشرط تطابق الرقم السرّي المُدخل مع

الرقم السري الموجود لدينا مسبقاً.

مثلاً لو أردنا كتابة الرمز $ عشر مرات عن طريق while ، سنكتبه بالطريقة التالية:

```
while (count <= 10)
{
      write $;
      count++ ;
}
```

نلاحظ أن while و for يحتوون على نفس الأجزاء ، ولكن أماكنها تختلف.

تبقَّى الجزء الأول؟ صحيح ، الجزء الأول يجب أن نكتبه قبل بداية أمر while ، لأننا لو كتبناه في داخلها ، سيصبح المتغير الذي يتحكم بعدد مرات التكرار count بنفس قيمته في كل دورة جديدة. فالمفروض يكون المثال السابق كالتالي:

```
var count=1;
while (count<=10)
{
      write $;
      count++ ;
}
```

وطبعاً كما ذكرت في مقدمة هذا الكتيِّب ، الأوامر يتم تنفيذها بالترتيب ، بمعنى أن البرنامج هذا سيقوم بكتابة علامة $ ثم سيزيد قيمة المتغير count.

نستنتج مما سبق ، أن البرنامج هذا:

```
for (count=1; count<=10; count++)
{
      write $;
}
```

والبرنامج هذا:

```
var count = 1;
while (count <= 10)
{
      write $;
      count++ ;
}
```

يقومون بنفس العمل ، ونتيجتهم ستكون واحدة.

توجد حالات ، استخدام أمر for فيها سيكون أسهل من while ، وحالات أخرى العكس ، فإذا فهمتم طريقة عمل كل أمر بشكل جيد وبعدما تبدأون في التطبيق والبرمجة الفعلية ، ستعرفون بأنفسكم ماهو الأمر الأنسب للاستعمال في الحالات المختلفة.

وأكرر: " بشكل عام ، أمر for سنستخدمه إذا كنّا سنكرر عمليةٍ ما بعدد مرات معروفة مسبقاً ، لكن لو كنّا نرغب مثلاً في أخذ رقم سري من المستخدم وسنقوم بتكرار إظهار نافذة الإدخال حتى يقوم بكتابة الرقم السري بشكل صحيح ، سنستخدم أمر while ، لأننا لا نريد تحديد عملية التكرار بعدد معينّ ، وإنما سنحدده بشرط تطابق الرقم السرّي المُدخل مع الرقم السري الموجود لدينا مسبقـاً."

3- if..else :

هذا الأمر يجعل البرنامج يختار بين حالات أو أشياء مختلفة ، مثال:

```
if (شرط أو مقارنة)
{
      إذا كانت نتيجة الشرط صحيحة سيتم تنفيذ هذا الأمر أو الأوامر
}
else
{
      إذا كانت نتيجة الشرط خاطئة سيتم تنفيذ هذا الأمر أو الأوامر
}
```

يحتوي أمر if على شرط أو مقارنة ، إذا كانت المقارنة صحيحة سيتم تنفيذ الأمر أو الأوامر الموجودة في محتوى if ، وإذا كان الشرط خاطئ ، سيتم تنفيذ الأمر أو الأوامر الموجودة في محتوى else.

نلاحظ هنا أن الأمر else لايحتوي على مقارنة أو شرط ، وهذا لأنه مرتبط بالمقارنة الموجودة مع if ، وأيضاً استعملنا الأقواس {المتعرجة} لمحتوى ل الأمرين ، حتى يستطيع البرنامج أن يعرف أين يبدأ محتوى كل أمر و أين ينتهي.

أمر else لانحتاج لوجوده دائماً ، وإذا لم نكتبه سيعمل البرنامج بشكل عادي وطبيعي ، فلذلك وجوده من عدمه يعتمد على هدفنا في البرمجة ، هل سنحتاج إليه أم لأ لتحقيق الهدف من البرنامج ، فمثلاً في بعض الحالات سنريد تنفيذ أمر في حالة معينة ، وإذا لم تتحقق هذه الحالة لا نريد البرنامج أن يقوم بشيء آخر.
ولكن لو أردنا كتابة else لابد أن نكون كتبنا قبله if ! فأمر else مرتبط بـ if ولا يمكنه أن يتخلّى عنه ، ولكن if غير مرتبط بـ else ، ويمكنه أن يكون وحيداً بدونه.

مثال:

```
if ( 10 < 20 )
{
     write True
}
else
{
     write False
}
```

هذا البرنامج سيكتب لنا كلمة True ، لأن الشرط صحيح ، لذلك سينفذ محتوى if.

ويمكننا أن نستعمل في برامجنا أكثر من if وأكثر من else بحيث تكون متداخلة مع بعض ، وتسمى nested if...else وسنرى مثال عليها في البرنامج رقم 1 في هذا الكتاب.

هكذا نكون انتهينا من شرح هذا القسم ، وأتمنّى أن يكون الشرح واضح وبسيط ، لأن البرمجة بمختلف لغاتها تعتمد على فهم هذه الأوامر وطريقة عملها. فكل لغات البرمجة توجد فيها if و for و while وكلّها تعمل بنفس الطريقة التي شرحتها.

6- جافاسكربت و HTML :

الأكواد البرمجية الخاصة بجافاسكربت يتم كتابتها في داخل أكواد وشفرات لغة HTML في أماكن مخصصة وبين شفرات محددة . كما نعلم ، الشفرات الأساسية الموجودة في أي ملف HTML هي:

```
<html>
      <head>
            <title> </title>
      </head>

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

شفرات وأكواد الجافا سكربت ممكن أن نكتبها في داخل <head> أو في داخل <body> .

الفرق بين المكانين هو كالتالي، المكان الأول <head> سيتم تنفيذ الأكواد اذا قمنا باستدعائها بأكواد خاصة في داخل شفرة <body> ، والمكان الثاني <body> سيتم تنفيذ الأكواد الموجودة فيه مباشرةً بمجرد وصول البرنامج (المتصفح) لها ، تذكرون بأن الأوامر يتم تنفيذها بالتسلسل ، صح؟

مصدر المعلومة: http://www.w3schools.com/js/js_whereto.asp

وشفرات وأكواد جافا سكربت تكتب في داخل هذا الكود الرئيسي:

```
<script language ="JavaScript">
هنا نكتب أوامر جافاسكربت
</script>
```

ووظيفتها هي أنها تخبر المتصفِّح بأن مابينها هي أكواد جافاسكربت وليست أكواد HTML.

فأكواد الجافاسكربت ستكون في أحد المكانين ، إما الأحمر أو الأزرق أو الإثنين معاً ، بحسب حاجتنا:

```
<html>
      <head>
            <title> </title>
            <script language ="JavaScript">
                  هنا نكتب أوامر جافاسكربت
            </script>
      </head>

      <body>
            <script language ="JavaScript">
                  هنا نكتب أوامر جافاسكربت
            </script>
      </body>
</html>
```

للأمانة حسب ماقرأت في المنهج كتابتهم لأكواد جافاسكربت كان في المكان الأحمر دائماً !

7- بعض الأوامر الشائعة :

1- document.write()

الأوامر عادةً تتكون من ثلاثة أقسام ، فلو أخذنا هذا الأمر على سبيل المثال:

```
document.write('Hello');
```

الجزء الأول document وهو يسمّى أوبجكت object

الجزء الثاني write وهو يسمى مثود method لهذا الأوبجكت

ودائماً يتم وصل الأوبجكت والمثود بنقطة . فقط ، أي أننا لانضع أي مسافة بينهم لأنها ستتسبب في عدم عمل البرنامج !

ويجب الانتباه إلى حجم الأحرف لأسماء الأوبجكت والمثود ، لأن لو كتبنا حرف كبير بدلاً من حرف صغير ، لن يعمل البرنامج !

الجزء الثالث يسمى argument وهو عبارة عن قوسين تحمل في داخلها parameters ، وفي مثالنا كلمة 'Hello' هي البارامتر.

هذا الأمر وظيفته هي الكتابة في المتصفِّح ، ففي المثال جعلناه يكتب كلمة Hello ، ووضعناها بين علامات 'التنصيص' لأنها string نريدها أن تظهر كما كتبناها بالضبط ، فلو أضفنا بعدها على سبيل المثال مسافة ، ستظهر هذه المسافة أيضاً في المتصفِّح.

لو كان لدينا برنامج ، يحتفظ باسم المستخدم في متغيِّر اسمه userName ، ومن ثم أردنا أن نكتب رسالة ترحيبية للمستخدم باسمه المخزَّن مسبقاً في البرنامج ، سنكتب الأمر التالي:

```
document.write('Hello ' + userName);
```

نلاحظ هنا أن اسم المتغيِّر userName لايوجد حوله علامات 'تنصيص' ! تذكَّروا الفرق بين المتغير العادي والـ string ، المتغير هو كلمة تدل على قيمة معيّنة ، فنحن لانريد أن نتعامل مع اسم المتغيِّر ، ولكن نريد أن نتعامل مع قيمته ، فلو وضعنا اسم المتغير هنا بين 'علامات التنصيص' ستكون النتيجة في المتصفِّح كالتالي:

```
Hello userName
```

لكن إذا لم نضع حوله علامات التنصيص ، سيعرف البرنامج بأن هذه الكلمة لها قيمة معينة ، ولذلك سيقوم بالبحث عن هذه القيمة ومن ثم سيستعملها بدلاً من اسم المتغيِّر userName. فلو كان هذا المتغيِّر يحمل اسم 'Learner' كقيمة له ، ستكون نتيجة الأمر على المتصفح هي:

```
Hello Learner
```

علامة + هنا وظيفتها هي الجمع بين الـ string واسم المتغير الذي نريد البرنامج أن يكتبه ، وإذا أردنا أن نضيف كلام آخر بعد المتغيِّر (اسم المستخدم) يجب أن نضع علامة + أخرى للدمج بينهم ، مثلاً لو أردنا إضافة علامة تعجب بعد اسم المستخدم

سنكتب:

```
document.write('Hello ' + userName + ' !');
```

وأيضاً أحب أن أنوِّه هنا على المسافتين الموجودة بعد كلمة Hello وقبل علامة التعجّب ، لأننا إذا لم نضع هذه المسافات ستتلاصق الكلمات وعلامة التعجّب مع بعضها البعض ، فيجب الانتباه لها.

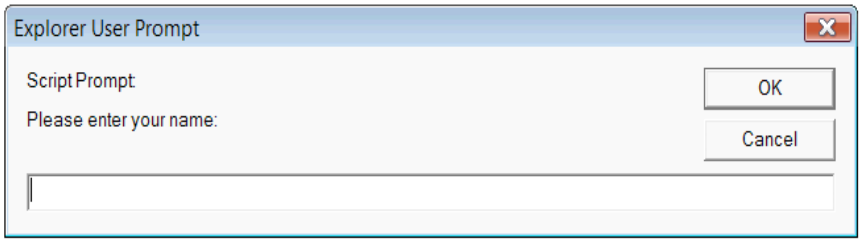
لو افترضنا بأن قيمة المتغيِّر userName الموجودة في البرنامج هي 'Learner' ، ستكون نتيجة الأمر الأخير هي:

```
Hello Learner !
```

window.prompt() -2

window هو الأوبجكت ، و prompt هو المثود لأوبجكت window

مثل الأمر السابق، يتكون من ثلاث أجزاء ، فائدة هذا الأمر بأنه يُظهِر نافذة صغيرة للمستخدم ، النافذة تحتوي على جزأين ، الأول هو رسالة (نكتبها نحن) ونريدها أن تظهر في النافذة ، والجزء الثاني سيكون مربع فارغ حتى يدخل المستخدم فيه قيمة معينة ، كما في الصورة التالية :



هذه النافذة هي نتيجة للأمر التالي:

```
window.prompt('Please enter your name:', ' ');
```

نلاحظ أن الجزء الثالث في الأمر يحتوي على جزأين كما ذكرت قبل قليل ، **الجزء الأول** خاص بالرسالة التي نريدها أن تظهر في النافذة ، وفي هذه الحالة الرسالة هي Please enter your name . **والجزء الثاني** وضعنا علامتي تنصيص فارغة وهي خاصة بالمربع أو المكان الذي سيكتب فيه المستخدم ، لو كتبنا شيء بين علامتي التنصيص مثل: اكتب هنا، ستظهر جملة اكتب هنا في المربع الأبيض. والجزأين يفصل بينهم فاصلة عادية ,

سؤال: كيف نأخذ القيمة التي سيدخلها المستخدم؟

جواب: عن طريق إنشاء متغير ، ونعطيه القيمة التي سيدخلها المستخدم.

سؤال: طيب كيف؟

جواب: تابع معي المثال التالي:
لو أردنا عمل برنامج يقوم بأخذ قيمة بالريال السعودي ويحولها إلى دولار أمريكي. في البداية سنقوم بتعريف متغيِّر ، وسنجعل اسمه money على سبيل المثال:

```
var money;
```

بعدها ، نعطي القيمة المدخلة عن طريق المستخدم للمتغير الذي أنشأناه بالشكل التالي:

```
money = window.prompt('please enter a value in S.R:', ' ');
```

هنا أحب أن أذكر نقطة مهمة، عندما نستعمل هذه الطريقة فإن جميع القيم التي سيدخلها المستخدم سيتم اعتبارها string ، ونحن أحياناً لانريد هذا الشيء لأننا سنحتاج في كثير من الأحيان إلى استخدام البيانات المدخلة من قبل المستخدم في معادلات رياضية وعمليات حسابية مختلفة ، مثل المثال الذي نطبقه الآن.
فإذا أردنا تحويل القيم المُدخلة من قبل المستخدم من string إلى متغيِّر عادي نستطيع أن نتعامل معه في العمليات الحسابية ، يجب علينا أن نستخدم الأمر parseFloat() ، واستخدامه يكون ببساطة كالتالي:

```
money = parseFloat(money);
```

يعني لو كانت القيمة التي أدخلها المستخدم هي 100 على سبيل المثال، ستكون القيمة الموجودة في المتغيِّر money هي '100' ، وبعدما نستعمل مثود parseFloat() كما في الأمر الأخير ، ستصبح قيمتها 100 ، أي أنها تحوّلت من string إلى متغيِّر عادي.

الآن القيمة الموجودة في المتغيِّر money قابلة للاستخدام في المعادلات الرياضية ، فنستطيع الآن أن نكتب معادلة رياضية لتحويل المبلغ من ريال سعودي إلى دولار أمريكي كالتالي:

```
money = money * 3.75;
```

أو يمكننا اختصار كتابتها كالتالي:

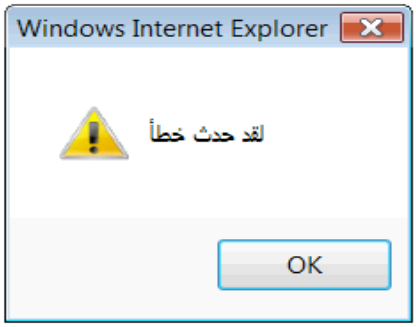
```
money *= 3.75;
```

الآن قيمة المتغير الجديدة ستكون نفس المبلغ الذي أدخله المستخدم ولكن بالدولار بدلاً من الريال.
نستطيع الآن أن نظهر نتيجة التحويل للمستخدم عن طريق الأمر:

```
document.write(money + ' $');
```

window.alert() -3

window هو الأوبجكت ، و alert هو مثود لأوبجكت window. وهو يستخدم لإظهار النافذة التالية:



الأمر الذي كتبته لإظهار هذه النافذة هو:

```
window.alert('لقد حدث خطأ');
```

وفائدته هي إظهار رسائل للمستخدم في حالات معينة لنخبره بأي شيء ، مثل حدوث خطأ أو أنه قام بإدخال قيمة غير صحيحة وغيرها من رسائل التنبيه المختلفة.

window.confirm() -4

window هو الأوبجكت ، و confirm هو مثود لأوبجكت window. وهو يستخدم لإظهار النافذة التالية:

الأمر الذي استعملته لإظهار هذه النافذة هو :

```
window.confirm('هل أنت متأكد؟');
```

إذا قام المستخدم هنا بالضغط على OK سيعود الأمر بقيمة True ، وإذا ضغط على Cancel سيعود الأمر بقيمة False.

طبعاً فائدة هذا الأمر كبيرة ولانستطيع الاستغناء عنه لأننا عن طريقه نستطيع أن نتأكد من أن المستخدم قام بعملية معينة ، يعني ممكن نستخدم أمر if بأنه إذا كانت قيمة هذه النافذة True ، نفذ أمر معينّ ، وإذا كانت False افعل شيء آخر.

Math.round() -5

Math هو الأوبجكت ، و round هو المثود. هذا الأمر وظيفته هي تقريب الرقم إلى أقرب رقم صحيح ، بمعنى لو كان لدينا على سبيل المثال رقم 9.3 ، عن طريق هذا الأمر نستطيع تقريبه إلى 9.

طبعاً عملية التقريب للتذكير تكون كالتالي:
إذا كان الرقم الذي بعد الفاصلة بين 0 و 4 سيتم تقريبه إلى الرقم الأصغر ..
وإذا كان الرقم الذي بعد الفاصلة بين 5 و 9 سيتم تقريبه إلى الرقم الأكبر ..

بمعنى:

```
Math.round(9.4) = 9
Math.round(9.0) = 9
Math.round(9.5) = 10
Math.round(9.9) = 10
```

فائدة هذا الأمر كبيرة ، وعلى سبيل المثال في عمليات القسمة التي تحدث في البرنامج (إن وجدت) تكون نتيجتها تحتوي على أعداد عشرية كثيرة ، على سبيل المثال لو قسمنا 10 على 3 ستكون النتيجة في البرنامج 3.33333333333334 وفي بعض الحالات نحن نريد البرنامج أن يعرض لنا الرقم الناتج عن العمليات الحسابية في المتصفح ، ولكن عرضه بهذا الشكل وبأرقام عشرية كبيرة غير جيد المنظر وغير مقبول للمستخدم ، لذلك نستعمل هذا الأمر للتقريب إلى أقرب رقم صحيح أو إلى عدد خانات عشرية نحدده نحن بإضافة بسيطة على الأمر حتى يصبح شكل الرقم المعروض مقبول لدى المستخدم.

الآن تعلمنا كيف نقرب الأرقام إلى أقرب رقم صحيح (بدون خانات عشرية) ، سننتقل للتقريب إلى عدد خانات عشرية محددة ..

أرجو التركيز في النقطة التالية:
لو أردنا عن طريق هذا الأمر أن نقرب رقم إلى خانتين عشرية فقط ، كيف نعملها بواسطة هذا الأمر ؟
مثلاً لو كان لدينا الرقم 9.456456 ونريد تقريبه إلى خانتين عشريتين ، ليصبح 9.46

أولاً: نضرب الرقم في 100:

```
9.456456 * 100 = 945.6456
```

ثانياً: نعمل تقريب لأقرب رقم صحيح:

```
Math.round(945.6456) = 946
```

أو يمكننا أن نجعل ما بداخل القوس هو نفس العملية الأساسية:

```
Math.round(9.456456 * 100) = 946
```

ثالثاً: نقسم الناتج على 100:

```
946 / 100 = 9.46
```

أو

```
Math.round(9.456456 * 100) / 100 = 9.46
```

وهذه هي النتيجة التي نريدها 9.46

مثال ثاني للتقريب إلى ثلاث خانات عشرية حتى تثبت المعلومة وتتضح أكثر ..
إذا أردنا تقريب الرقم 9.368921 إلى ثلاث خانات عشرية حتى يصبح 9.369

أولاً: نضرب الرقم في 1000 (بنفس عدد الخانات التي نريدها)

```
9.368921 * 1000 = 9368.921
```

ثانياً: نعمل تقريب لأقرب رقم صحيح:

```
Math.round(9368.921) = 9369
```

أو

```
Math.round(9.368921 * 1000) = 9369
```

ثالثاً: نقسم الناتج على 1000

```
9369 / 1000 = 9.369
```

أو

```
Math.round(9.368921 * 1000) / 1000 = 9.369
```

وهذه هي النتيجة التي نريدها 9.369

وهكذا لباقي الحالات ..

8- كتابة التعليقات والملاحظات :

التعليقات هي شيء أساسي في أي لغة برمجة لأن لها فوائد كثيرة ، منها أنها توضِّح لنا الفائدة والغرض من كتابة أمر معين وبالتالي يستطيع من يقرأ الأمر أن يعرف سببه ويفهمه بسهولة ، وأيضاً يساعد المبرمج الذي كتب الأمر البرمجي - إذا عاد للاطلاع عليه ليصحح بعض الأخطاء بعد مرور فترة زمنية طويلة - في تذكر سبب كتابته والفائدة منه ، وبالتالي يستطيع التعديل وتصحيح الأخطاء بسرعة و سهولة. وأيضاً يمكننا عن طريقها معرفة مكان الأخطاء في البرنامج بسهولة أكثر ، وسأعود لهذه النقطة بعد قليل.

كتابة التعليقات والملاحظات يكون بأحد طريقتين:

الأولى اسمها تعليق سطر واحد one line comment:

```
// هذا تعليق ولن يظهر في الصفحة
```

الثانية اسمها تعليق لعدة أسطر multi-line comment:

```
/*
وهذا تعليق آخر لن يظهر في الصفحة ولا يمكن رؤيته إلا في ملف الأوامر
أو مايسمى بـ Source code
*/
```

طبعاً هذه التعليقات يجب أن تكون من ضمن أكواد <script language='JavaScript'> و </script> ، لأنها لو كانت خارجها ، سيعتبرها المتصفح من ضمن لغة HTML وبالتالي سيقوم بكتابتها في الصفحة كما هي ، لأن لغة HTML لايوجد فيها وسم أو شفرة مشابهة لها.

وطبعاً التعليقات لاتظهر أبداً في المتصفحات ، وفائدتها الرئيسية هي للمبرمج نفسه وبقية المبرمجين الذي سيقرأون الكود البرمجي.

طيب الآن نعود لنقطة فائدتها في تسهيل اكتشاف الأخطاء ، كما نعلم في لغة جافاسكربت اذا وجِد خطأ بسيط جداً ، ستظهر لنا صفحة فارغة لا تحتوي على أي شيء ، وهذا شيء يجلب الإحباط للأسف خاصة إذا لم نستطع معرفة الخطأ بسرعة ، ولذلك يمكننا استخدام التعليقات في اكتشاف مكان الخطأ لتقليل الوقت المستهلك في اكتشاف و تصحيح الأخطاء ، خاصة الصغيرة منها.

في أحد البرامج الصغيرة الذي كنت أعمل عليه ، استخدمت فيه عدة أوامر ومن ضمنها أمر if ، وطبعاً المبرمج إذا تعوَّد على لغة البرمجة ستصبح كتابته سريعة جداً مثل كتابة اللغة العادية ، فاحتمال وجود الأخطاء الإملائية كبير.

المهم أني بعد ما انتهيت من كتابة البرنامج وقمت بتشغيله لأرى النتيجة لم يظهر شيء ، قلت عادي أكيد خطأ بسيط كالعادة ، سأقوم بتصحيحه الآن لتجربة البرنامج. جلست ما يقارب الأربعين دقيقة ولم أستطع معرفة مكان الخطأ!!!

بعدها فكّرت بطلب مساعدة التعليقات ، فقمت بتقسيم البرنامج الصغير إلى ثلاثة أقسام:
الأول جعلته خاص بالمتغيرات وتعاريفها الموجودة في البداية.
الثاني خاص بالمعادلات وأوامر الاختيار والتكرار وهي في منتصف البرنامج.
الثالث خاص بعرض نتائج البرنامج في المتصفِّح وهي في آخر البرنامج.

فقمت الآن بوضع تعليقات حول الجزأين الثاني والثالث ، ووضعت أمر كتابة في المتصفِّح في الجزء الأول ، جعلته يكتب كلمة OK ، بعدها قمت بتشغيل البرنامج وظهرت لي كلمة OK في المتصفِّح ، وهذا يعني أن الجزء الأول لاتوجد به مشاكل.
بعدها أزلت التعليقات من الجزأ الأخير لأتأكد منه ، وبعد التجربة اتّضح لي بأن المشكلة ليست فيه.

الآن عرفت بأن المشكلة موجودة في الجزء الثاني ، يعني في أوامر الاختيار والتكرار والمعادلات الحسابية ، فقمت بتقسيم هذه المنطقة إلى عدة أجزاء حتى أختبرها جزءاً جزءاً ، إلى أن استطعت - ولله الحمد - اكتشاف الخطأ الذي أخذ مني قرابة الساعة أو أكثر لاكتشافه.

هل تعلمون ماذا كان الخطأ؟
في أحد أوامر `if` ، قمت بكتابة القوس المحيط بالشرط هكذا: `{` ، والمفترض أن يكون: `(`
خطأ تافه ، ولكن الأخطاء في البرمجة – خاصة بدون استعمال برامج اكتشاف وتحديد الأخطاء – لاترحم ، ولا تفرِّق بين الخطأ الكبير والصغير ، وستتسبب في ضياع وقت كبير إذا لم نعرف كيف نكتشفها بسرعة. فلو استخدمنا التعليقات بذكاء ستحل لنا مشاكل كثيرة في تصحيح الأخطاء.

9- أمثلة تطبيقية :

البرنامج رقم 1: تصنيف البيض لمقاسات بحسب الوزن

هذا المثال هو السؤال الرابع في TMA02 لسنة 2008 ، السؤال يشمل عدة نقاط هامّة ولذلك استخدمته كمثال.

السؤال: اكتب برنامج بلغة جافاسكربت يقوم بالآتي:

أولاً: يظهر نافذة للمستخدم تطلب منه إدخال وزن البيض.

ثانياً: يقوم بكتابة الوزن - الذي أدخله المستخدم - في الصفحة للتأكيد.

ثالثاً: يقوم بتصنيف مقاس البيض بحسب العوامل التالية:

- إذا كان الوزن أقل من 53 جرام ، يُعتبر صغير.
- إذا كان الوزن يساوي أو أكبر من 53 و أقل من 63 جرام ، يكون وسط.
- إذا كان الوزن يساوي أو أكبر من 63 و أقل من 73 ، يكون كبير.
- إذا كان الوزن يساوي أو أكبر من 73 ، يكون كبير جداً.

السؤال يحتوي على ثلاث فقرات مساعدة للحل ، سنمشي عليها كلها.

الفقرة الأولى:

المطلوب هنا ، هو التخطيط للبرنامج عن طريق كتابة البرنامج باللغة العادية (الإنجليزية) وليست لغة البرمجة ، لأنها تعتبر عادة ممتازة للمبرمجين في تنفيذ أي برنامج لأنها تساعد على ترتيب الأفكار.

طريقة الحل تكون كالتالي:

declare a variable for the weight تعريف متغير

get a value from the user and assign it to the variable أخذ وزن من المستخدم وتعيينها للمتغير الذي عرفناه

display a confirmation message with the value إظهار رسالة تأكيد بالوزن الذي أدخله المستخدم

هكذا نكون انتهينا من أول خطوتين وهي أخذ قيمة من المستخدم ووضعها في متغير ، وإظهار الوزن المُدخل في الشاشة للتأكيد.

ننتقل للخطوة الثالثة ، وهي التي عليها الكلام ..

نبدأ بأول تصنيف وهو إذا كان الوزن أقل من 53 يكتب لنا أن البيض صغير:

```
if ( the weight is smaller than 53)
        display a message (The egg size is small)
```

انتيهنا الآن من اذا كان البيض صغير ، فننتقل للي بعده ، وهو اذا كان الوزن أكبر من أو يساوي 53 و أصغر من 63 يكتب لنا ان مقاس البيض وسط:

```
else
    if (the weight is larger than or equal to 53 AND less than 63)
        display the message (The egg size is medium)
```

انتهينا من الحالة الثانية ، ننتقل للحالة الثالثة ، وهي اذا كان الوزن أكبر من أو يساوي 63 وأقل من 73 يكتب لنا ان المقاس كبير ، وهي نفس طريقة الحالة الثانية:

```
    else
        if ( the weight is larger than or equal to 63 AND less than 73)
            display the message (The egg size is large)
```

الآن نجي للحالة الرابعة والأخيرة ، وممكن تحل بطريقتين:

1- أن نضيف أمر else ونكتب داخلها أمر كتابة في الصفحة بأن الحجم كبير جداً ، وسنكتبها كالتالي:

```
        else
            display the massage (The egg size is very large)
```

2- أن نكتب else وبعدها if لإذا كان الوزن أكبر من أو يساوي 73 اكتب في الشاشة أن المقاس كبير جداً ، وسنكتبها هكذا:

```
        else
            if ( the weight is larger than or equal to 73 )
                display the massage (The egg size is very large)
```

الطريقة الأولى معناها أن أي وزن غير مطابق للحالات الثلاثة الأولى راح يتم اعتباره كبير جداً. والطريقة الثانية نفس الأولى ولكننا نحدد الأرقام فيها. وكل الطريقتين صحيحة ان شاء الله ، وستعطي نفس النتيجة بالضبط.

الفقرة الثانية:

المطلوب هنا هو تحويل الأوامر الموجودة في الفقرة السابقة إلى كود حقيقي وقابل للاستعمال ..

أولاً سنجهز مكان أكواد الجافاسكربت ، يعني سنقوم بعمل أكواد HTML الرئيسية في البداية ، وسنضع أكواد الجافا السكربت في مكان الاستفهامات:

```
<html>
     <head>
          <title> </title>
          <script language='JavaScript'>???????</script>
     </head>
     <body>
     </body>
</html>
```

في السؤال طلب منا وضع عنوان للصفحة باسم Egg sizes ، والعنوان يتم كتابته بين شفرات <title> .
ولابد من تنسيق الكود والأوامر بشكل جيد ، حتى يكون الكود سهل القراءة للمدرِّس أو لغيره. المقصود بالتنسيق هو زيادة المحاذاة عن الطرف الأيسر باستخدام زر tabs في الكيبورد ، حتى نبيِّن الأوامر والشفرات ونميّزها عن بعض.

سنفترض أن القيمة المدخلة من المستخدم ، ستكون أرقام أو مدخلات صحيحة ، فلا داعي أن نضع أوامر للتأكد من أن المستخدم لم يقم مثلاً بكتابة أحرف أو غيرها.

الآن نرجع لحلنا في الفقرة الأولى ، وسنمشي عليه أمراً أمراً ، حتى نحول جميع الأوامر إلى أكواد فعليَّة.
أولاً ، عملية تعريف المتغير الذي سنحفظ فيه مدخلات المستخدم:

declare a variable for the weight

سنسمي المتغيِّر weight ، ولذلك سيكون الأمر:

```
var weight;
```

انتهينا من الأمر الأول: تعريف المتغير.

الأمر الثاني هو عملية أخذ قيمة من المستخدم وتعيين القيمة للمتغير weight :

```
weight = window.prompt('Please enter the weight of the egg in grams.', '');
```

الآن الكمبيوتر إذا أخذ أي قيمة من المستخدم راح يعتبرها string مثل ماشرحنا في الأقسام السابقة ، لذلك سنقوم بتحويل القيمة إلى متغيِّر نستطيع التعامل معه في المعادلات والمقارنات عن طريق parseFloat :

```
weight = parseFloat(weight);
```

ننتقل للخطوة التي بعدها ، وهي كتابة أمر تأكيد بالوزن الذي أدخله المستخدم ، وسيكون مثل "وزن البيض هو ... جرام"
مكان النقط ... سنضع اسم المتغير weight الذي يحمل الوزن الذي أدخله المستخدم. فيكون الأمر كالتالي:

```
document.write('The weight of the egg is ' + weight + ' grams.<br/>');
```

طبعاً أجزاء هذه الأكواد تم شرحها في الأقسام السابقة ، لذلك لن أعيد شرحها مرة أخرى.

ننتقل لعملية البرمجة الفعلية ، وهو كتابة أوامر المقارنات بين الأوزان وإظهار رسالة معينة لكل وزن..

نبدأ بأول تصنيف وهو إذا كان الوزن أقل من 53 يكتب لنا أن البيض صغير:

if (the weight is smaller than 53)

display a message (The egg size is small)

هذا هو التخطيط الذي كتبناه في حل الفقرة السابقة ، ولتحويله إلى كود جافاسكربت نكتب:

```
if (weight < 53) {
      document.write('The egg size is small.<br/>');
}
```

القوسين {المتعرجة} التي كتبناها ضرورية لتعريف البرنامج متى تبدأ الأوامر المتعلقة بـ if ومتى تنتهي. وهذا الأمر سيتم تنفيذه فقط إذا كان الوزن الذي أدخله المستخدم أقل من 53.

ننتقل للحالة الثانية ، اذا كان وزن البيض أكبر من أو يساوي 53 و أقل من 63:

else

if (the weight is larger than or equal to 53 AND less than 63)

display the message (The egg size is medium)

هذا هو التخطيط الذي كتبناه في حل الفقرة السابقة ، ولتحويله إلى كود جافاسكربت نكتب:

```
else {
      if ( weight >= 53 && weight < 63 ) {
            document.write('The egg size is medium.<br/>');
      }
```

هذا الأمر سيتم تنفيذه في حالة كان الوزن أكبر من أو يساوي 53 وفي نفس الوقت أقل من 63.

ونتَّبع نفس الطريقة للحالتين الثالثة والرابعة:

```
      else {
            if ( weight >= 63 && weight < 73 ) {
                  document.write('The egg size is large.<br/>');
            }
            else {
                  document.write('The egg size is very large.<br/>');
            }
      }
}
```

بالنسبة للأقواس {المتعرِّجة} :

نفتح القوس بعدما نكتب الأمر (سواء if أو else أو for أو while) ، ونغلقه بعد نهاية الأوامر التي تتعلق بالأمر.

ففي الكود الخاص بالحالة الثانية ، نلاحظ أنني فتحت قوس بعد أمر else ولكني أم أقم بإغلاقه! السبب هو أن كل الأوامر المتبقية ستكون من ضمن هذه الـ else ، فلذلك قمت بإغلاقه بعد كتابة الحالتين الثالثة والرابعة.

فكروا فيها بالمنطق .. في البداية قلنا: "إذا كان وزن البيض أقل من 53" ، والآن كتبنا: "غير ذلك" ، هذا يعني بأن "غير ذلك" هذه ستحتوي على كل الأوزان التي تزيد عن 53 . وللتوضيح أكثر سأضع الأقواس على التخطيط بالفقرة الأولى ، وسألون فقط أقواس else المرتبطة ببعض بنفس اللون ، لأن أقواس if في برنامجنا هذا تنتهي بعده مباشرة:

```
if ( the weight is smaller than 53) {
        display a message (The egg size is small)
}
else {
        if (the weight is larger than or equal to 53 AND less than 63) {
                display the message (The egg size is medium)
        }
        else {
                if ( the weight is larger than or equal to 63 AND less than 73) {
                        display the message (The egg size is large)
                }
                else {
                        display the message (The egg size is very large)
                }
        }
}
```

النتيجة النهائية للبرنامج ستكون بالشكل التالي:

```
<html>
<head>
<title>Egg sizes</title>
<script language='JavaScript'>
var weight;
weight = window.prompt('Please enter the weight of the egg in grams.', '');
weight = parseFloat(weight);
document.write('The weight of the egg is ' + weight + ' grams.<br/>');

if (weight < 53) {
      document.write('The egg size is small.<br/>');
```

```
}
else {
      if ( weight >= 53 && weight < 63 ) {
            document.write('The egg size is medium.<br/>');
      }
      else  {
            if ( weight >= 63 && weight < 73 ) {
                  document.write('The egg size is large. <br/>');
            }
            else {
                  document.write('The egg size is very large. <br/>');
            }
      }
}
</script>
</head>
<body>
</body>
</html>
```

في النهاية ، نأخذ هذا الكود وننسخه في برنامج تحرير نصوص مثل notepad ونحفظ الملف بأي اسم ، ولكن الصيغة أو النوع يجب أن تكون html . يعني الملف النهائي يجب أن يكون نوعه أو امتداده html وليس doc أو txt مثلاً!
ولتشغيل البرنامج نفتح الملف عن طريق متصفِّح الانترنت ، سواء انترنت إكسبلورر أو فايرفوكس أو غيرها وستظهر لنا نتيجة البرنامج.

الفقرة الثالثة:

يقول كيف نتأكد من أن البرنامج يعمل بشكل صحيح ولا توجد به أخطاء؟

حتَّى نتأكد من أن البرنامج صحيح ولا توجد به أخطاء ، يجب أن نختبر جميع الحالات الموجودة في البرنامج ، وبرنامجنا هذا فيه أربع حالات ، وهي: إما صغير إذا كان < 53 ، أو وسط إذا كان >=53 و <63 ، أو كبير إذا كان >=63 و <73 ، أو كبير جداً إذا كان >73.

الأرقام ممكن تختلف من شخص لآخر ، ولكن هذه الأرقام - من وجهة نظري - هي الأفضل والأضمن:
رقم 52 ، حتى نختبر الحد الأعلى للحالة الأولى وهي الحجم الصغير.
رقم 53 ، حتى نتأكَّد من الحد الأعلى للحالة الأولى ، وأيضاً نختبر الحد الأدنى للحالة الثانية وهي الحجم المتوسط.
رقم 63 ، حتّى نتأكد من الحدّ الأعلى للحالة الثانية ، وأيضاً نختبر الحد الأدنى للحالة الثالثة المتعلقة بالحجم الكبير.
رقم 73 ، حتّى نتأكد من الحدّ الأعلى للحالة الثالثة ، وأيضاً نختبر الحد الأدنى للحالة الرابعة المتعلقة بالحجم الكبير جداً.

البرنامج رقم 2: لعبة التخمين

هذا سؤال أتى لنا في الاختبار النهائي:

قم بتخزين رقم في البرنامج بين 1 و 1000 واجعل المستخدم يحاول أن يقوم بتخمينه أو معرفته عن طريق إدخال رقم بين 1 و 1000 ..

إذا أدخل المستخدم رقم أكبر من الرقم المخزّن أظهر له رسالة تخبره بأنه فوق الرقم الصحيح ودعه يحاول مرة أخرى.

إذا أدخل المستخدم رقم أصغر من الرقم المخزّن أظهر له رسالة تخبره بأنه تحت الرقم الصحيح ودعه يحاول مرة أخرى.

وفي النهاية اذا أدخل الرقم الصحيح سنكتب له بأنه استطاع تخمين الرقم الصحيح بعد عدد المحاولات التي قام بها.

طريقة الحل لأي سؤال في البرمجة هو فهم السؤال ومن ثم تقسيمه وتحليله إلى خطوات ، طبعاً التقسيم والتحليل سيكون في عقلنا لأن في الاختبار لايوجد مكان كافي للكتابة !

أولاً: سنعمل ثلاثة متغيرات ، المتغير الأول سنسميه ourNumber وسنضع له قيمة نختارها ولنفرض انها 501 ، والمتغير الثاني سنسميه userNumber وسنجعله يحمل القيمة التي يدخلها المستخدم ، والثالث سنسميه count وستكون مهمته حساب عدد المحاولات التي قام بها المستخدم.

ثانياً: سنقارن بين الرقم الذي وضعناه والرقم الذي أدخله المستخدم:

إذا كان الرقم الذي أدخله أكبر ، سنقوم بإظهار رسالة تخبره بأنه فوق الرقم الصحيح وسنجعله يحاول مرة أخرى.

إذا كان الرقم الذي أدخله أصغر ، سنقوم بإظهار رسالة تخبره بأنه تحت الرقم الصحيح وسنجعله يحاول مرة أخرى.

ثالثاً: إذا استطاع تخمين الرقم الصحيح:

سنكتب رسالة له نخبره بأنه استطاع تخمين الرقم الصحيح بعد عدد المحاولات التي قام بها.

نحول الخطوات إلى أكواد ..

أولاً:

```
var ourNumber, userNumber, count;
ourNumber = 501;
userNumber = window.prompt('Enter a number between 1 and 1000', '');
userNumber = parseFloat(userNumber);
count = 1;
```

السطر الأول ، عرَّفنا ثلاثة متغيرات.

السطر الثاني ، وضعنا رقم 501 في المتغير الخاص بنا.

السطر الثالث ، جعلنا المتغيِّر الخاص بالمستخدم يأخذ القيمة من المستخدم.

السطر الرابع ، قمنا بتحويل الـ string إلى variable يمكن التعامل معه في المقارنات والمعادلات.

السطر الخامس ، وضعنا قيمة 1 للمتغير الخاص بعد عدد المحاولات ، وبدأ بـ 1 لأن المستخدم بدأ فعلياً بأول محاولة في السطر الثالث.

ثانياً:

سنستعمل لهذا الجزء أمر while ، وسنجعل الشّرط: إذا لم يتطابق الرقم الذي أدخله المستخدم مع الرقم 501 :

```
while (ourNumber != userNumber) {

}
```

إذا لم يتطابق الرقمين ستكون لدينا حالتين: إما أن يكون رقم المستخدم أكبر 501 أو يكون أصغر من 501 ، لذلك سنضيف في داخل محتوى while أوامر if و else وسنجعل if لإذا كان رقم المستخدم أكبر من 501 وسنعرض له رسالة تخبره بأن رقمه أكبر من الرقم الصحيح ، و else ستكون لحالة إذا كان رقم المستخدم أصغر من 501 وسنعرض له الرسالة المناسبة :

```
while (ourNumber != userNumber) {
     if (userNumber > ourNumber) {
          window.alert('You are above the right number!');
     }
     else {
          window.alert('You are below the right number!')
     }
}
```

بعدما قمنا بإخبار المستخدم بأنه إما فوق أو تحت الرقم الصحيح ، نريد إضافة أمر حتى يحاول مرة أخرى ، وسنضيف تحته مباشرة أمر آخر لزيادة المتغيِّر الخاص بعد عدد المحاولات:

```
while (ourNumber != userNumber) {
     if (userNumber > ourNumber) {
          window.alert('You are above the right number!');
     }
     else {
          window.alert('You are below the right number!')
     }

     userNumber = window.prompt('Try again :', '');
     userNumber = parseFloat(userNumber);
     count = count + 1;
}
```

طبعاً بداخل أمر while يوجد خيارين فقط ، وهي إما أن يكون الرقم أصغر من 501 أو يكون أكبر منه ، لايمكن أن يكون يساويه لأن شرط while هو أنهم غير متساويين ، فبالتالي لو كان الرقمين متساويين لن يتم تنفيذ محتوى while.

تبقّى لنا الجزء الثالث والأخير ، وهو إذا قام المستخدم بإدخال الرقم الصحيح ، وبالطبع لو كان الرقم المدخل يساوي الرقم 501 لن يتم تنفيذ while وإنمّا سيتم تنفيذ مابعدها من أوامر إن وجدت ، فسنكتب هذه الجزئية بعد while كالتالي:

```
document.write('You guessed the right number after ' + count + ' tries!');
```

سيقوم البرنامج بكتابة لقد قمت بتخمين الرقم الصحيح بعد عدد المحاولات المخزنة في المتغير count.
الآن نجمع الأجزاء الثلاثة مع بعض:

```
var ourNumber, userNumber, count;
ourNumber = 501;
userNumber = window.prompt('Enter a number between 1 and 1000', '');
userNumber = parseFloat(userNumber);
count = 1;

while ( ourNumber != userNumber)
{
      if (ourNumber < userNumber)
      {
            window.alert('You are above the right number!');
      }
      else
      {
            window.alert('You are below the right number!');
      }

      userNumber = window.prompt('Try again :', '');
      userNumber = parseFloat(userNumber);
      count = count + 1;
}
document.write('You guessed the number after ' + count + ' tries!');
```

وهذا هو البرنامج المطلوب بشكل كامل إن شاء الله.

البرنامج رقم 3: إيجاد القاسم المشترك الأكبر بين رقمين

سؤال أتى لنا أيضاً في الاختبار النهائي ..

السؤال يقول: اعمل برنامج يجعل المستخدم يدخل رقمين ، ويقوم البرنامج بإيجاد العامل المشترك الأكبر بين الرقمين ويعرضه في الصفحة.

المشكلة الرئيسية هي في طريقة إيجاد العامل المشترك الأكبر لأنها تحتاج وقت للتفكير والتجريب حتى نستطيع إيجادها لأول مرة.

العامل المشترك الأكبر بين رقمين المقصود فيه هو أكبر رقم ممكن أن نقسم عليه الرقمين ويكون الناتج عدد صحيح.

مثلاً لو أردنا العامل المشترك الأكبر ل 6 و 3 الناتج هو 3 لأنه أكبر رقم نستطيع أن نقسم عليه الرقمين ونحصل على رقم صحيح.

ولو أردنا العامل المشترك الأكبر ل 50 و 40 سنبحث عن أكبر رقم ممكن أن نقسم عليه هذين الرقمين ونحصل على رقم صحيح في الحالتين، والإجابة في هذه الحالة ستكون 10 لأنه أكبر رقم نستطيع أن نقسم عليه الرقمين والنتيجة ستكون عدد صحيح.

ولو أردنا العامل المشترك الأكبر للرقمين 20 و 15 سيكون الناتج 5 لنفس السبب.

طيب المشكلة الآن هي كيف نحول هذا الكلام إلى أكواد وأوامر برمجية؟!

الحل ممكن يكون بأكثر من طريقة وسأشرح لكم طريقتي في حله (طبعاً بعد الاختبار حليته :D)

طريقتي هي أنني أنشأت متغير غير المتغيريين الذين أدخل المستخدم فيهم أرقاماً ، وجعلت المتغير هذا يأخذ قيمة أصغر رقم فيهم ، يعني إذا كان الرقم الأول أصغر من الثاني ستصبح قيمة المتغير الذي عرفته تساوي الرقم الأول والعكس.

وبعدها وضعت شرط وهو : إذا كان باقي قسمة الرقم الأول على المتغير لا تساوي صفر أو إذا كان باقي قسمة الرقم الثاني على المتغير لاتساوي صفر .. وفي المحتوى كتبت أمر يقوم بإنقاص قيمة هذا المتغير بواحد.

بافتراض أن الرقم الأول موجود في متغيِّر اسمه x والرقم الثاني في متغيِّر اسمه y ، والمتغيِّر الذي سيحمل قيمة الرقم الأصغر سأسمّيه GCD ، اختصاراً لـ Greatest Common Divisor بمعنى القاسم المشترك الأكبر ، سيكون الشرط كالتالي:

```
while ( x % GCD != 0 || y % GCD != 0 )
{
     GCD = GCD – 1;
}
```

بصراحة الشرط صعب بالنسبة للمبتدئين لذلك حاولوا أن تركزوا في شرحي له الآن:

القاسم المشترك الأكبر بين رقمين هو إذا كان باقي قسمة الرقمين عليه تساوي صفر ، مثل الأمثلة التي طبقناها قبل قليل ، لذلك وضعنا الشرط بأنه إذا كانت باقي قسمة أي من الرقمين على المتغير لا تساوي صفر معناها بأننا لم نصل بعد للقاسم المشترك الأكبر للرقمين ونحتاج لتنقيص قيمته بواحد حتى تصل قيمته إلى رقم يحقق الشرط الموجود وهو رقم القاسم المشترك الأكبر.

أولاً: سنقوم بتعريف ثلاث متغيرات، الأول سيحمل الرقم الأول الذي سيدخله المستخدم ، والثاني سيحمل الرقم الثاني الذي سيدخله المستخدم ، والثالث سنجعله يحمل قيمة الرقم الأقل منهم.
حتى نختصر الشرح والوقت ، سنحذف فقرة إدخال الأرقام من المستخدم ، وسندخل رقمين من لدينا مباشرةً للتسهيل.

سنفرض أن الرقمين هم 40 و 30 ، وسنضعهم في المتغيرين x و y على التوالي:

```
var x = 40;
var y = 30;
```

سنضع المتغير الثالث ليحمل قيمة العدد الأصغر ، وهو y في هذه الحالة:

```
var GCD = y;
```

الآن نكتب أمر while مع الشّرط الذي شرحته قبل قليل:

```
while ( x % GCD != 0 || y % GCD != 0 ) {
     GCD = GCD – 1;
}
```

اذا استبدلنا المتغيرات بقيمها (للتوضيح) سنحصل على:

```
while ( 40 % 30 != 0 || 30 % 30 != 0 ) {
     GCD = GCD – 1;
}
```

بمعنى اذا كان باقي قسمة 40 على 30 لا تساوي صفر أو اذا كانت باقي قسمة 30 على 30 لا تساوي صفر قم بإنقاص قيمة المتغير GCD بواحد.

طيب البعض يمكن يتساءل: لماذا لم نستعمل && بدلاً من || ؟
لو نتذكر طريقة عمل كل واحد منهم سنعرف السبب ..
لو استخدمنا && ، لن يتم تنفيذ محتوى while ، إلا إذا كان كل الطرفين - الموجودين في الشرط – صحيحين ، أي أن كل الرقمين باقي قسمتهم على المتغيّر لا تساوي صفر.
يعني في المثال الأخير ، لو استعملنا && في الشرط ، لن يدخل في محتوى while ، لأن 0!=40%30 ستكون True ، ولكن 0!=30%30 ستكون False ، لذلك لن يتم تنفيذ محتوى while.

ولو استخدمنا || ، سيتم تنفيذ محتوى while في جميع الحالات ، إلا إذا كان كل الطرفين False ، يعني كل الطرفين باقي قسمتهم على المتغيّر تساوي صفر ، وهذا معناه بأن المتغير أصبح الآن يحمل قيمة العامل المشترك الأكبر.
أرجو أن أكون وفّقت في توضيح الفكرة و المعلومة. :)

بما أننا الآن عرفنا كيف نوجد القاسم المشترك الأكبر ، الباقي بسيط ان شاء الله وسأجعله واجب لكم :) ، أكملوا حل السؤال وإذا واجهتكم مشكلة اكتبوها ، وسنتعاون في حلِّها سويةً إن شاء الله.

ولمن يريد أن يزيد استيعابه لهذه المسألة ، أرفقت مع ملف الشّرح برنامج لحلها في ثلاث حالات: **الأولى** إذا كان الرقم الأول أكبر من الرقم الثاني ، **والثانية** إذا كان الرقم الأول أصغر من الثاني ، **والثالثة** إذا كان الرقمين متساويين. وقمت بحلّ الحالة الأولى عن طريق أمر `for` ، والحالة الثانية عن طريق أمر `while` للتنويع :).

String methods -10 :

الآن نحن نعلم بأن 95 يعتبر متغيِّر عادي ، و '95' أو 'Learner' يعتبرون string. في البرمجة توجد methods تعطينا إمكانيات للتحكّم و التعامل مع الـ strings ، وفي المنهج تم شرح ثلاثة منها وسأقوم بتفصيلها الآن.

قبل أن أبدأ بها ، أحب أن أوضّح نقطة بسيطة: الآن لو افترضنا بأن لدينا string يحتوي على كلمة Learner ، هذا الـ string –وأي string غيره – يكون له طول ، الطول يكون عدد الأحرف الموجودة فيه ، يعني على مثالنا سيكون طول الـ string هو 7 لأن عدد الأحرف كلها 7.
وأيضاً كل حرف في الـ string سيكون له مركز أو ترتيب ، الترتيب عادي ولكن الاختلاف فقط هو في بداية الترتيب ، لأن أول حرف سيكون ترتيبه 0 وليس 1 كما هو متعارف عليه في الواقع.

يعني بالنسبة لحرف L في كلمة Learner سيكون ترتيبه أو مركزه هو 0 ، وحرف n مثلاً سيكون ترتيبه 4.
وفي الجدول التالي توضيح أكثر:

String	L	e	a	r	n	e	r
مركز الحرف	0	1	2	3	4	5	6
الطول				7			

charAt() -1 :

هذا المثود المفترض أن نضع بين القوسين التابعة له رقم ، وهو سيعطينا الحرف المقابل لهذا الرقم. لو ترجمت الأمر باللغة العادية سيكون كأننا نقول: أعطنا الحرف الموجود في هذا المركز.

لنفرض أننا قمنا بتعريف المتغير التالي ونوعه string :

```
var userName = 'Learner';
```

لو كتبنا مثلاً:

```
userName.charAt(0);
```

ستكون نتيجة هذا الأمر L

نلاحظ أن طريقة كتابة الـ method تكون عن طريق كتابة اسم المتغيِّر الذي نوعه string ، ثم نقطة . ثم كتبنا اسم المثود.
يجب الانتباه إلى حرف A الكبير الموجود في المثود ، لأننا لو كتبنا بدلاً منه حرف a الصغير لن يعمل البرنامج!
لو وضعنا هذا الأمر الأخير في داخل أمر الكتابة في المتصفح كالتالي:

```
document.write(userName.charAt(0));
```

سيكتب لنا في المتصفح L ، ولو استبدلنا الصفر برقم 6 سيكتب لنا r .

فائدة هذا الأمر هو اننا نستطيع أن نعرف مما يتكون الـ string ، مثلاً لو عملنا برنامج ومن ضمنه مكان لتسجيل اسم مستخدم وكلمة سر خاصة بالمستخدم ، نستطيع عن طريق هذا الأمر أن نتأكد مثلاً من أن أول خانة لا تكون رقم ، أو أن تتكون كلمة السر من أحرف وأرقام مختلطة ببعض ، وغيرها ...

2- length :

هذا المثود مهمّته هي أن يعطينا ويحدد لنا طول الـ string ، وهو لايحتاج إلى argument مثل باقي الـ methods ، فهو يستطيع العمل بدونها.

لو كان لدينا هذا الـ string في البرنامج:

```
var userName = 'Learner';
```

ثم كتبنا هذا الأمر لنعرف طول الكلمة:

```
userName.length;
```

ستكون النتيجة 7 ..

طيب لو كان الـ string الذي لدينا كالتالي:

```
var sentence = 'Hello World';
```

ثم كتبنا:

```
sentence.length;
```

ستكون النتيجة 11 لأن المسافة محسوبة أيضاً مع الأحرف! وأي شيء سيكون متواجد في الـ string سواءً كان رمزاً أو حرفاً أو رقماً أو أي شيء ، سيتم احتسابه أيضاً ومعاملته كالحروف.

فائدة هذا الأمر تكون عندما نأخذ كلمة من المستخدم ، عن طريقه نستطيع أن نعرف طول الكلمة ، لو أردنا مثلاً عمل برنامج لتسجيل اسم مستخدم و كلمة سر خاصة بالمستخدم ، نستطيع عن طريق هذا الأمر أن نحدد أدنى طول لكلمة السر ، مثل أن لاتقل عن 6 خانات.

3- indexOf() :

هذا الأمر نقوم بكتابة حرف في داخل الأقواس وسيقوم بتحديد مكان أو مركز الحرف. فلو كان لدينا الـ string التالي:

```
var userName = 'Learner';
```

ثم قمنا بكتابة الأمر التالي:

```
userName.indexOf('L');
```

ستكون النتيجة 0

ولو غيرنا القيمة كالتالي:

```
userName.indexOf('n');
```

ستكون النتيجة 4

ولو غيرناها لـ :

```
userName.indexOf('e');
```

نلاحظ انه يوجد حرفين مشابهين لهذا الحرف في الـ string ، والنتيجة ستكون مكان أول حرف وهو 1.

طيب لو كان الـ string الذي لدينا:

```
var sentence = 'Hello World';
```

وكتبنا هذا الأمر:

```
sentence.indexOf('World');
```

سيعطينا البرنامج مكان أول حرف في هذه الكلمة وهو 6.

مثال تطبيقي على ماسبق:

لو جاء لنا سؤال يقول: اعمل برنامج يقوم بأخذ اسم المستخدم ، ثم يقوم البرنامج بعرض كل حرف من أحرف الاسم في سطر مستقل.

أولاً: سنقوم بتعريف متغير ونجعل المستخدم يكتب اسمه ، وسنحفظ الاسم في هذا المتغير:

```
var userName = window.prompt('Please enter your name:', '');
```

ثانياً: عن طريق أمر for سنجعل البرنامج يطبع كل حرف من هذا الاسم في سطر مستقل:

قبلما أبدأ في الخطوة الثانية ، أريدكم أول شيء أن تفهموا كيف سنطبع كل حرف في سطر منفصل ..

طبعاً لتحديد وللتعامل مع كل حرف سنستعمل مثود charAt()

فإذا أردنا طباعة كل حرف سنستعمل هذه الأوامر:

charAt(0)
charAt(1)
charAt(2)
charAt(3)
إلى أن ينتهي عدد الأحرف

لكننا بالطبع لن نكتبها بهذا بالشكل لسببين ، الأول هو أننا لانعرف كم عدد الأحرف التي أدخلها المستخدم ، والثاني هو أن كتابتها بهذا الشكل لو كنّا نعرف كم عدد الأحرف سيحتاج لوقت وجهد كبيرة.

لذلك سنستغل قوة أمر for ، وسنضع فيه متغير يبدأ من الصفر ثم يزيد بواحد في كل دورة حتى يتم طباعة كل الأحرف ، هذا المتغيِّر سنستخدمه بدلاً من الأرقام في داخل مثود charAt() ، وسيكون الشرط عبارة عن: قيمة المتغيَّر تكون أصغر من طول الاسم ، لأنه كما تعلّمنا: مركز آخر حرف في الـ string يكون أقل من طول الـ string بواحد ، فلو كان طول الـ string هو 7 ، سيكون مركز آخر حرف فيها هو 6.

```
for (var i=0; i<userName.length; i=i+1) {
}
```

مرة أخرى: i ستحل محل الأرقام في داخل أمر charAt() ، وكما رأينا قبل قليل ، فائدة أمر length هي أنه يعطينا عدد الأحرف الموجودة في الـ string. مثلاً لو كان عدد أحرف الـ string يساوي 7 ، المتغيِّر i ستزيد قيمته سبع مرات بواحد ، حتى يصل إلى رقم 6 وهو مركز آخر حرف في string طوله أو عدد أحرفه 7.

وبالطبع لايمكننا أن نجعل المتغيِّر i يبدأ من رقم 1 مثلاً! لأننا هكذا سنتجاوز الحرف الأول في الـ string وسنبدأ من الحرف الثاني.

طيب ، الآن سنكتب أمر طباعة كل حرف في سطر مستقل في داخل محتوى أمر for :

```
for (var i=0; i<userName.length; i=i+1) {
     document.write(userName.charAt(i) + '<br/>');
}
```

سأشرح أول دورتين أو عمليتي تكرار حتى تتضح لكم الصورة أكثر بافتراض أن الـ string طوله 7 ..

أولاً ستبدأ i وقيمتها 0 ، ثم ستتم عملية المقارنة 7 > 0 ، والنتيجة صحيحة لذلك سيدخل في المحتوى ، وسيقوم بتنفيذ هذا الأمر:

```
document.write(userName.charAt(0) + '<br/>');
```

ثم سيذهب للجزء الثالث وسيزيد قيمة i بواحد لتصبح قيمتها تساوي 1

ثم سيرجع للقسم الثاني ويتحقق من الشرط وسيكون الشرط صحيح ، لذلك سيدخل للمحتوى مرة أخرى وسينفذ هذا الأمر:

```
document.write(userName.charAt(1) + '<br/>');
```

هكذا نكون طبعنا الحرف الثاني في السطر الثاني ، وسيستمر البرنامج بالعمل حتى يقوم بكتابة كل الأحرف السبعة ، كل حرف في سطر مستقل. وهذا هو الكود بعد تجميعه:

```
var userName = window.prompt('Please enter your name:', '');

for (var i=0; i<userName.length; i=i+1)
{
     document.write(userName.charAt(i) + '<br/>');
}
```

هذه الفكرة والطريقة مهمة جداً في عالم البرمجة وتساعد على معرفة قوة أمر for وفوائده ، لذلك حاول أن تفهمها بشكل جيِّد.

11- المصفوفات Arrays :

المصفوفات أو مايسمّى بـ Arrays هي متغيّرات تسمح لنا بتخزين أكثر من قيمة فيها ، وكل قيمة نستطيع الحصول عليها من خلال مركزها أو ترتيبها في المصفوفة. مثلاً لو كنا نريد أن نحفظ تكاليف الأشهر للسنة في برنامج لنقوم بأخذ متوسط التكاليف الشهرية ، بدلاً من أن نقوم بحفظ تكلفة كل شهر في متغير خاص ، نستطيع أن ننشئ مصفوفة واحدة حجمها 12 (عدد الأشهر) ثم نضع تكاليف الأشهر في المصفوفة.

توجد أكثر من طريقة لإنشاء المصفوفات وسنستعرضها جميعها إن شاء الله. لو افترضنا بأننا نريد أن ننشئ مصفوفة تحتوي على أسماء أشهر السنة الميلادية:

الطريقة الأولى:

```
var monthsArray = new Array(12);
monthsArray[0] = 'January';
monthsArray[1] = 'February';
...
```

حتى نصل إلى آخر شهر

```
monthsArray[11] = 'December';
```

هنا قمنا بإنشاء مصفوفة اسمها monthsArray . وفائدة new Array هي إخبار البرنامج بأن المتغير الذي أنشأناه هو لمصفوفة وليس متغيِّر عادي (مثل طريقة تعريف الـ string). رقم 12 المكتوب بين القوسين يدل على أن حجم المصفوفة (يعني عدد عناصرها) هو 12.
هذه الطريقة استخدامها قليل ، لأنه توجد طرق أخرى أسهل وأسرع في الكتابة.

طبعاً أول عنصر في المصفوفة رقمه أو ترتيبه يكون 0 مثل رقم أول حرف في الـ string ، لذلك أول عنصر في مصفوفتنا هذه سيكون رقمه 0 وآخر عنصر سيكون رقمه 11 .

الطريقة الثانية:

```
var monthsArray = 'January','February','March','April','May','June','July'
                  ,'August','September','October','November','December'];
```

نلاحظ ان هذه الطريقة أسهل بكثير من الطريقة الأولى وأيضاً نلاحظ بأن أسماء الأشهر مكتوبة بين علامتي تنصيص (للدلالة على أنها string) ، وتفصل بينهم فاصلة عادية , ووضع العناصر بين القوسين [ ] يخبر البرنامج بأنهم عناصر مصفوفة ، يعني بدلاً من كتابة new Array كما في الطريقة الأولى ، قمنا بإضافة العناصر مباشرة بين القوسين [ ] للتعريف بالمصفوفة.

الطريقة الثالثة:

هذه الطريقة تشبه الأولى ، ولكن تختلف عنها بأننا لن نحدد حجم المصفوفة فيها، بمعنى سنترك مابين القوسين فارغ بدون رقم. في كثير من الأحيان عندما نستعمل المصفوفات نكون لانعلم كم عدد العناصر التي سنضعها في المصفوفة إما لكثرتها أو لأنها غير معلومة لدينا في وقت كتابة البرنامج أو لأننا نريد المستخدم أن يكتبها ، وفي هذه الحالة سنستعمل هذه الطريقة

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

وبعده بإمكاننا كتابة عناصر المصفوفة بنفس الطريقة الأولى ، وبإمكاننا أن نجعل المستخدم يكتب العناصر أيضاً عن طريق كتابة اسم المصفوفة ومركز العنصر وبعدها علامة يساوي وبعدها أمر window.prompt() ، مثل:

```
monthsArray[0] = window.prompt('Enter the first month name:', '');
```

وعندما يدخل المستخدم اسم الشهر الأول سيتم تخزينه في المركز 0 في هذه المصفوفة.

طيب الآن عرفنا كيف نعرِّف وننشئ المصفوفة ، وكيف نضع قيم فيها، لذلك سننتقل إلى بعض الـ methods الهامّة والتي تساعدنا في التعامل مع المصفوفات.

1- indexOf() :

نفس الأمر الذي استعملناه مع الـ string ، وكانت وظيفته هي تحديد مكان الحرف في الـ string . في المصفوفات وظيفته هي تحديد مركز مابين القوسين في داخل المصفوفة. على سبيل المثال لو كان لدينا مصفوفة أشهر السنة:

```
var monthsArray = 'January','February','March','April','May','June','July'
                  ,'August','September','October','November','December'];
```

إذا استعملنا الأمر التالي:

```
document.write(monthsArray.indexOf('January'));
```

سيكتب لنا في المتصفح رقم 0 ، أي أن هذا الأمر قام بتحديد مركز أو ترتيب هذا الشهر في المصفوفة ، وهذه الطريقة مهمة في بعض الحالات ، مثل إذا أردنا البحث في عناصر المصفوفة وسيتم التطرق لها لاحقاً إن شاء الله.

2- length :

أيضاً نفس الأمر الذي استعملناه مع الـ strings ، ووظيفته هنا هي تحديد حجم أو عدد عناصر المصفوفة. مثلاً مصفوفة الأشهر:

```
var monthsArray = 'January','February','March','April','May','June','July'
                  ,'August','September','October','November','December'];
```

لو كتبنا هذا الأمر:

```
document.write(monthsArray.length);
```

سيكتب في المتصفح رقم 12 لأن عدد عناصر المصفوفة هو 12 .

سؤال: كيف نعمل برنامج فيه مصفوفة لأسماء أشهر السنة ، ونجعل المستخدم يدخل أسماء الأشهر ؟
جواب: تتذكرون الطريقة التي استعملناها لطباعة كل حرف في الـ string في سطر مستقل بواسطة أمر for ؟
سنستعمل نفس الفكرة بالضبط ..

أولاً ننشئ مصفوفة ونسميها ونحدد حجمها 12 لأن عدد الأشهر 12

```
var monthsArray = new Array(12);
```

ثانياً نعمل أمر for ، لإظهار رسالة لأخذ اسم الشهر وتخزينها في المكان المناسب في المصفوفة ، وسنعرِّف متغيِّر i حتى يتحكّم بعدد مرات التكرار:

```
for (var i=0; i<monthsArray.length;i++) // i=i+1 is the same as i+=1 and i++
{
     monthsArray[i] = window.prompt('Enter the month name:', '');
}
```

في الدورة الأولى في هذا الأمر ستكون قيمة i = 0 ، أي أن الأمر الموجود في محتوى for سيكون كالتالي:

```
monthsArray[0] = window.prompt('Enter the month name:', '');
```

وفي الدورة الثانية أو التكرار الثاني ستكون قيمة i = 1 ، وسيكون الأمر الموجود في محتوى for كالتالي:

```
monthsArray[1] = window.prompt('Enter the month name:', '');
```

وسيستمر بالتكرار حتى تصبح i=12 ، وعندها سيصبح الشرط الموجود خاطئ ولن يتم تنفيذ المحتوى ، لأن monthsArray.length يساوي 12 ... وبالتالي سيكون الشرط 12 < 12 ، والنتيجة False.

لو وضعنا بدلاً من علامة (أصغر من) علامة (أصغر من أو يساوي) سيكون الشرط صحيح وسيكون الأمر الموجود في المحتوى كالتالي:

```
monthsArray[12] = window.prompt('Enter the month name:', '');
```

وهذا معناه انه طلب من المستخدم كتابة اسم شهر رقم 13 (لأننا بدأنا من 0) وهذا يعني أن النهاية لابد أن تكون 11 حتى تكتمل الاثني عشر شهراً ، وإذا أكملنا ل 12 مثل ماعملنا في آخر أمر ، سيكون معناه بأننا طلبنا كتابة اسم الشهر رقم 13 !

هذا الكلام هو لزيادة توضيح فائدة أمر for لمثل هذه الأشياء.

للتلخيص هذا جدول يلخِّص الـ methods الخاصة بالمصفوفات وطريقة عملها:

var monthsArray = [	'Jan'	'Feb'	'Mar'	'Apr'	'May'	'June'	'July'	'Aug'	'Sep'	'Oct'	'Nov'	'Dec'
monthsArray.indexOf()	0	1	2	3	4	5	6	7	8	9	10	11
monthsArray.length	12											

12- الدَّوال Functions :

الدَّوال أو الـ functions هي عبارة عن أكواد نكتبها لعمل شيء أو مهمّة معينة ، مثل تقريب رقم لخانتين عشرية أو لأي عملية أخرى. الدّوال فائدتها هي انها تقوم بتجزئة الأوامر البرمجية الكبيرة والكثيرة إلى أجزاء صغيرة يسهل قراءتها والتّعامل معها ، وفائدتها الرئيسية هي أننا نكتبها مرة واحدة ، ونستطيع أن نستخدمها في البرنامج عدة مرات عن طريق استدعاء الدّالة. يعني مثلاً إذا أردنا أن نكتب أكواد عدة مرّات لتقريب الأرقام في برنامجنا إلى خانتين عشرية ، نستطيع أن نكتب دالّة مهمّتها تقريب الأرقام إلى خانتين عشرية ، وكلما احتجناها نقوم باستدعائها بأمر قصير ، بدلاً من أن نقوم بكتابة أكواد و أوامر التقريب أكثر من مرة.

توجد دوال معرفة في الجافاسكربت تلقائياً مثل دالّة `parseFloat` ، والتي تقوم بتحويل المتغيِّر من نوع `string` إلى متغيّر عادي ، وتوجد دوال نحن نقوم بإنشائها ، وسنتحدَّث عنها في هذا القسم إن شاء اللّه.

البرنامج رقم 4: دالّة لتقريب الأرقام إلى خانتين عشرية

لو كان لدينا البرنامج البسيط التالي:

```
var numOne = 7;
var numTwo = 3;
var divideOneOnTwo = numOne / numTwo;
var divideTwoOnOne = numTwo / numOne;

divideOneOnTwo = Math.round(divideOneOnTwo * 100) / 100;
divideTwoOnOne = Math.round(divideTwoOnOne * 100) / 100;
```

البرنامج عبارة عن تعريف متغيرين: الأول قيمته 7 والثاني قيمته 3 ، وبعدها عرفنا متغيرين: الأول عبارة عن قسمة قيمة أول متغير على ثاني متغير ، والثاني عبارة عن قسمة ثاني متغير على أول متغير.

وبعدها قمنا بتقريب قيمة المتغيرين الجديدين إلى خانتين عشرية عن طريق أمر `Math.round()` ، طبعاً طريقة التقريب تمّ شرحها بالتفصيل عندما شرحت أمر `Math.round()` في قسم "بعض الأوامر الشائعة".

طيب الآن بدل ما أكتب أوامر تقريب الرقم إلى خانتين عشرية في كل مرة أحتاج إلى تقريب رقم ، سأقوم بكتابة دالّة خاصة بهذه العملية ، وسأستدعيها كلّما احتجت إليها في البرنامج. سأسمّي الدّالة `twoDPs` ، اختصاراً لـ Two Decimal Places :

```
function twoDPs(anyNumber)
{
     return Math.round(anyNumber * 100) / 100;
}
```

بالنسبة لأول كلمة function ، هي كلمة محجوزة وتدل على وجود دالّة.
ثاني كلمة twoDPs ، هي اسم الدّالة.
بالنسبة لكلمة anyNumber الموجودة بين القوسين هي بمثابة دليل ، أي أن أي رقم سيأتي في داخل القوسين ، سيتم وضعه في مكان هذه الكلمة الموجود في أمر Math.round() .
مثلاً لو استدعينا الدالة ، ووضعنا في داخل القوسين هذا الرقم 9.76543 ، سيتم التعويض بهذا الرقم في محتوى الدالة لتصبح:

```
return Math.round(9.76543 * 100) / 100;
```

وطبعاً يمكننا تسميته بأي اسم نرغب فيه ولكن يفضّل بأن يدل الاسم على القيمة التي من المفترض أن يحتوي عليها. إذا لم تتضح فائدة هذه الكلمة تابعوا معي وستتضح إن شاء الله بعد قليل.

بالنسبة لكلمة return الموجودة في محتوى الدّالة ، هي تعني بأننا إذا قمنا باستدعاء هذه الدالة فنحن نريد منها أن تعيد لنا قيمة ، أي أننا في مثالنا نريدها أن تعيد لنا في المكان الذي استدعينا فيه الدالة الرقم بعد تقريبه لخانتين عشرية بدلاً من الخانات الكثيرة الموجودة حالياً.

الآن عملنا الدّالة ، لو افترضنا بأننا نريد استعمالها في نفس البرنامج الذي كتبناه في البداية ، سيتحوّل البرنامج إلى:

```
var numOne = 7;
var numTwo = 3;
var divideOneOnTwo = numOne / numTwo;
var divideTwoOnOne = numTwo / numOne;

divideOneOnTwo = twoDPs(divideOneOnTwo);
divideTwoOnOne = twoDPs(divideTwoOnOne);
```

فقط نقوم بكتابة اسم الدالة وفي داخلها القيمة التي نريد تقريبها. دعونا نتتبّع الخطوات التي سيعملها البرنامج ابتداءً من أول أمر ، وانتهاءً بآخر أمر لتتضح لكم الصورة:
السطر الأول والثاني: أنشأنا متغيرين الأول قيمته 7 والثاني قيمته 3
السطر الثالث: أنشأنا متغير قيمته تساوي قسمة 7 على 3 ، يعني:

```
divideOneOnTwo = 2.3333333333
```

السطر الرابع: أنشأنا متغير قيمته تساوي قسمة 3 على 7 ، يعني:

```
divideTwoOnOne = 0.4285714286
```

السطر الخامس: طلبنا تغيير قيمة المتغيِّر divideOneOnTwo عن طريق دالّة twoDPs ، أي أننا نريد تقريب قيمته السابقة إلى خانتين عشرية. وسيكون الأمر كالتالي:

```
divideOneOnTwo = twoDPs(2.3333333333);
```

في السطر الخامس ، قمنا باستدعاء دالّة ، لذلك سنذهب الآن للدالّة التي استدعيناها قبل تنفيذ مافي السطر الخامس. وستكون الدّالة بهذا الشكل:

```
function twoDPs(2.3333333333)
{
     return Math.round( 2.3333333333 * 100) / 100;
}
```

هل اتّضحت لكم فائدة كلمة anyNumber الآن ؟

الآن الدالّة ستعود للمكان الذي استدعيناها فيه - لأننا طلبنا منها ذلك عن طريق أمر return - بنتيجة العمليات الحسابية التي قامت بها ، وستضع النتيجة في مكانها. يعني السطر الخامس سيتحوّل بعد عمل الدّالة إلى:

```
divideOneOnTwo = 2.33 ;
```

الآن سينتقل البرنامج للسطر السادس ، وسيفعل نفس مافعله مع السطر الذي سبقه. وسيكون شكل الأمر كالتالي:

```
divideTwoOnOne = twoDPs(0.4285714286);
```

بعدها سيذهب البرنامج إلى الدالة ومعه الرقم 0.4285714286 ، وثم سيعود بالنتيجة الحاصلة من عمل الدالة وهو الرقم 0.43 ، وسيضعه في المكان الذي استدعينا فيه الدالة:

```
divideTwoOnOne = 0.43 ;
```

يعني لو أضفنا بعد آخر أمر في هذا البرنامج أوامر لكتابة هذين المتغيرين في المتصفح ، ستكون النواتج 2.33 و 0.43 على التوالي.

والكود الكامل للبرنامج سيكون:

```
function twoDPs(anyNumber)
{
     return Math.round(anyNumber * 100) / 100;
}

var numOne = 7;
var numTwo = 3;
var divideOneOnTwo = numOne / numTwo;
```

```
var divideTwoOnOne = numTwo / numOne;

document.write(divideOneOnTwo + ' and ' + divideTwoOnOne + ' before
rounding.<br/>');

divideOneOnTwo = twoDPs(divideOneOnTwo);
divideTwoOnOne = twoDPs(divideTwoOnOne);

document.write(divideOneOnTwo + ' and ' + divideTwoOnOne + ' after
rounding.);
```

طبعاً هنا أضفت أمرين لكتابة قيم المتغيرين قبل التقريب وبعد التقريب حتى تتّضح لكم الصورة أكثر. وستكون نتيجة هذا البرنامج المكتوبة في الصفحة هي:

```
2.3333333333333335 and 0.42857142857142855 before rounding.
2.33 and 0.43 after rounding.
```

البرنامج رقم 5: دالّة لحساب المساحة بالمتر المربّع

لو أردنا عمل برنامج يأخذ من المستخدم طول وعرض لمكان معين، وسيقوم البرنامج بحساب مساحة المكان عن طريق دالّة. وللتذكير المساحة تساوي الطول ضرب العرض!

أولاً ننشئ متغيرين وندع قيمتهم ليدخلها المستخدم:

```
var length = window.prompt('Enter the length', '');
var width = window.prompt('Enter the width', '');
```

ثانياً نعمل دالة لتقوم بعملية ضرب الطول في العرض وتعيد النتيجة لنا:

```
function getArea(length, width)
{
     return length * width;
}
```

نلاحظ هنا أن بين القوسين يوجد اسمين، بمعنى حتّى تعمل هذه الدالة ، يجب أن نرسل لها رقمين ، وهي في حالتنا طول وعرض. ومن الممكن أن يكون داخل القوسين عدد لانهائي من القيم ، فهو شيء يعتمد على حاجتنا وغرضنا من كتابة الدّالة.

ثالثاً نضع أمر لكتابة النتيجة بالشكل التالي:

```
document.write('The area is ' + getArea(length, width) );
```

نلاحظ هنا ان الدّالة استدعيناها في داخل أمر الطباعة وهو شيء عادي ، وأحببت أن أوضح لكم فيه أن استدعاء الدالة ممكن أن يكون في أي مكان في البرنامج.
والـ arguments التي وضعناها بين القوسين ، هي قيم الطول والعرض التي أدخلها المستخدم.

البرنامج سيكون كالتالي:

```
function getArea(length, width)
{
     return length * width;
}

var length = window.prompt('Enter the length', '');
var width = window.prompt('Enter the width', '');

document.write('The area is ' + getArea(length, width) );
```

طبعاً الدّوال يتم كتابتها غالباً بين وسمي <head> ، بعكس أمر document.write مثلاً ، الذي من المفترض أن نكتبه بين وسمي <body> ، وإنشاء المتغيرات وتعريفها ممكن يكون في أي من المكانين.

طيب نمشي مع برنامجنا الأخير خطوةً خطوةً ، على افتراض أن المستخدم أدخل رقم 10 للطول و 7 للعرض:

```
length = 10
 width = 7
```

سيذهب البرنامج الآن إلى أمر الكتابة في المتصفح وسيكتب في المتصفح:

```
The area is
```

ثم سيجد الدالة getArea وفيها قيمتين 10 و 7 على التوالي. سيذهب البرنامج الآن للدالة التي استدعيناها بالرقمين 10 و 7 لتصبح بالشكل التالي:

```
function getArea(10, 7)
{
     return 10 * 7;
}
```

ثم سيعود بنتيجة عمل الدالة وهو رقم 70 إلى أمر الكتابة في المتصفح الذي توقف عنده ، وسيكمل الكتابة وسيكتب 70 بجانب الكلام الذي كتبه مسبقاً لتكون نتيجة الكتابة النهائية:

```
The area is 70
```

إن شاء الله تكون وضحت لكم فوائد الدّوال وطرق استخدامها.

البرنامج رقم 6: حساب متوسِّط التكاليف الشهرية

لو أردنا عمل برنامج يقوم بأخذ التكاليف للاثني عشر شهراً من المستخدم ، ومن ثم يقوم بعرضها في المتصفح ، بكتابة كل شهر وبجانبه تكلفته في سطر مستقل ، وفي النهاية يكتب لنا متوسط التكاليف لهذه الأشهر.
للتذكير: متوسط التكاليف يساوي مجموع التكاليف تقسيم عددها (عدد الأشهر).

سنمشي خطوة خطوة ، أولاً سنقوم بإنشاء مصفوفة لتحمل التكاليف الشهرية و حجمها 12 :

```
var costsArray = new Array(12);
```

الآن سنجعل المستخدم يقوم بإدخال التكاليف في هذه المصفوفة باستخدام أمر for :

```
for ( var i = 0 ; i < costsArray.length ; i++ ) {
     costsArray[i] = window.prompt('Enter the month cost', '');
}
```

هذا الأمر ناقص .. هل تذكرون بأن أي قيمة يقوم بإدخالها المستخدم تكون عبارة عن string ؟
إذن ، لابد أن نقوم بإضافة أمر parseFloat حتى نستطيع التعامل مع الأرقام المُدخلة ، كالتالي:

```
costsArray[i] = parseFloat(window.prompt('Enter the month cost', ''));
```

الآن نفذنا أول خطوة ، وستظهر عن طريقها رسالة 12 مرة للمستخدم ، ليدخل فيها التكاليف الشهرية.
الآن الرسالة التي ستظهر للمستخدم في كل مرة ستكون: Enter the month cost ، بمعنى أدخل تكاليف الشهر. لو افترضنا بأن المستخدم قام باستقبال مكالمة هاتفية أثناء إدخاله للتكاليف ، هل تعتقدون بأنه سيتذكر ماهو آخر شهر أدخل تكاليفه؟
غالباً لن يتذكر ، والبرنامج لايوجد فيه شيء يذكّره باسم أو رقم الشهر الذي توقف عنده.

لذلك سنقوم بتعديل بسيط على الكود ، حتى يظهر رقم الشهر في الرسالة. التعديل بسيط جداً ، سنقوم بإضافة المتغير i في الرسالة ، ولكننا سنزيد قيمته بواحد ، لأننا لا نريده أن يقول للمستخدم أدخل تكاليف الشهر 0 ! ، نريده أن يقول 1 لأن المستخدم غالباً لن يفهم هذه الأرقام والأساسيات البرمجية.
لذلك سنغير أمر الطباعة ليصبح بالشكل التالي:

```
costsArray[i] = window.prompt('Enter the monthly cost for month
                                number ' + (i+1), '');
```

لاحظوا أني وضعت i+1 في داخل قوس ، وهذا يجعل البرنامج يجمع قيمتهم قبل أن يقوم بإظهار الرسالة. وإذا لم نضع القوسين ، سيقوم البرنامج بإظهار قيمة i ، وبعده رقم 1 ، فهو سيعتقد بأن علامة الزائد هنا هي للجمع بين متغيرات مختلفة

وليست العملية الرياضية. بمعنى لو جعلناهم بدون قوسين ، ستكون أول رسالة على الشكل التالي:

```
Enter the monthly cost for month number 01
```

والرسالة الثانية ستكون:

```
Enter the monthly cost for month number 11
```

بمعنى أنه قام بكتابة قيم i ومن ثم كتب رقم 1. ولكن اذا وضعناهم داخل قوسين سيتم جمع قيمتهم أولاً ثم سيكتب المجموع ، وهو 1 في المرة الأولى و 2 في المرة الثانية.

طيب لو أردنا أن نستبدل الأرقام بأسماء الأشهر، كيف طريقتها؟ نفس الفكرة ، ولكن سنحتاج إلى كتابة مصفوفة تحتوي على أسماء الأشهر:

```
var monthsArray = ['January','February','March','April','May','June','July'
                ,'August','September','October','November','December'];
```

وسنقوم بالتعديل على أمر إظهار النافذة ليصبح بالشكل التالي:

```
costsArray[i] = parseFloat(window.prompt('Enter the monthly cost for ' +
                monthsArray[i], ''));
```

للتوضيح ، في أول عمليّة تكرار ، ستكون قيمة i = 0 ، وسيكون المحتوى كالتالي:

```
costsArray[0] = parseFloat(window.prompt('Enter the monthly cost for ' +
                monthsArray[0], ''));
```

والرسالة التي ستظهر في هذا الأمر هي:

```
Enter the monthly cost for January
```

الآن انتهينا من أمر إظهار رسالة للمستخدم بشكل منسّق ومرتّب ليقوم بإدخال التكاليف الشهرية في المصفوفة. وللتذكير هذا ماقمنا بعمله حتى الآن:

```
var costsArray = new Array(12);
var monthsArray = ['January','February','March','April','May','June','July'
                ,'August','September','October','November','December'];

for ( var i = 0 ; i < costsArray.length ; i++ )
{
      costsArray[i] = parseFloat(window.prompt('Enter the monthly cost for '
                      + monthsArray[i], ''));
}
```

الآن نريد أن نقوم بكتابة اسم كل شهر وتكلفته في سطر منفصل ، ببساطة سنقوم بإضافة أمر الكتابة في داخل محتوى for بالشكل التالي:

```
document.write(monthsArray[i] + ' : ' + costsArray[i] + '<br/>');
```

أعتقد أن الأمر واضح إذا كنّا فهمنا طريقة استعمال المتغيِّر i بهذا الشكل. سيتغير الآن محتوى for ليصبح:

```
for ( var i = 0 ; i < costsArray.length ; i++ )
{
      costsArray[i] = parseFloat(window.prompt('Enter the monthly cost for '
                        + monthsArray[i], ''));

      document.write(monthsArray[i] + ' : ' + costsArray[i] + '<br/>');
}
```

الأمر الأول سيأخذ قيمة أول عنصر في المصفوفة وهي تكلفة أول شهر. والأمر الثاني سيقوم بكتابة اسم أول شهر وبعده نقطتين وبعدها قيمة أول عنصر في المصفوفة وهي تكلفة الشهر الأول.

طبعاً ترتيب الأوامر بهذا الشكل ضروري جداً، لأننا لو كتبنا الأمر الثاني قبل الأمر الأول ، سنكون طلبنا من البرنامج أن يكتب في المتصفّح تكلفة أول شهر قبل أن يدخلها المستخدم ، وبالطبع مكان التكلفة في المصفوفة سيكون فارغ ، ولذلك ستظهر لنا كلمة undefined في مكان التكلفة.

تبقى لنا الآن عملية حساب المتوسط للتكاليف الشهرية ، وكما نعرف المتوسط سيساوي مجموع التكاليف على عددها ، يعني سنحتاج الآن إلى إضافة أمر يقوم بجمع التكاليف الشهرية. لذلك سننشئ متغيِّر جديد - ليحسب مجموع التكاليف - باسم sum وستكون قيمته الأوليّة 0 .

```
var sum = 0;
```

طيب الآن كيف نستطيع إضافة هذا المتغيِّر في داخل محتوى for حتى يجمع كل تكلفة جديدة يدخلها المستخدم؟
للتبسيط ، نريد أن تكون قيمة هذا المتغيِّر في كل تكرار جديد تساوي قيمتها السابقة زائد التكلفة التي أدخلها المستخدم.
يعني لو افترضنا أن المستخدم أدخل 1500 كأول تكلفة ، نريد sum تساوي 0 + 1500 .
ولو أدخل 2000 للتكلفة الثانية ، تكون sum تساوي 1500 + 2000 .
الأمر بسيط ولا يختلف عن الفكرة الأساسية لأمر for ، وسيكون بالشكل التالي:

```
sum = sum + costsArray[i] ;
```

الأكواد التي كتبناها حتى الآن هي كالتالي:

```
var costsArray = new Array(12);
var monthsArray = ['January','February','March','April','May','June','July'
                ,'August','September','October','November','December'];

var sum = 0;

for ( var i = 0 ; i < costsArray.length ; i++ )
{
     costsArray[i] = parseFloat(window.prompt('Enter the monthly cost for '
                     + monthsArray[i], ''));

     document.write(monthsArray[i] + ' , ' + costsArray[i] + '<br/>');

     sum = sum + costsArray[i] ;
}
```

الآن تبقا لنا المتوسط ، وسنقوم بعمل دالّة للقيام بالمهمة وسنستدعيها في أمر الكتابة في المتصفح:

```
function getAverage(total, number)
{
     return total / number;
}
```

سمّينا الداة getAverage ، ووضعنا لها قيمتين لن تعمل بدونهم ، الأولى هي المجموع والثانية هي العدد. طبعاً المجموع الآن أصبح جاهزاً في المتغيِّر sum ، وبالنسبة للعدد ، نستطيع إمّا أن نكتب رقم 12 عند استدعاء الدّالة ، أو نستطيع استخدام خاصية length لمصفوفة التكاليف.

يعني أمر كتابة المتوسط للتكاليف الشهرية سيكون بالشكل التالي:

```
document.write('The monthly costs average is ' + getAverage(sum,
                costsArray.length));
```

أين سنكتب هذا الأمر الأخير؟ في داخل محتوى for ؟ هل نريده أن يتنفَّذ أكثر من مرّة ؟ أم نريده أن يتنفّذ مرة واحدة فقط ؟ طبعاً نريد تنفيذه مرة واحدة فقط ، وذلك في نهاية البرنامج ، ولذلك سنكتبه بعد أمر for .

والكود الكامل سيكون بهذا الشكل:

```
function getAverage(total, number)
{
     return total / number;
}

var costsArray = new Array(12);
```

```
var monthsArray = ['January','February','March','April','May','June','July'
                ,'August','September','October','November','December'];

var sum = 0;

for ( var i = 0 ; i < costsArray.length ; i++ )
{
     costsArray[i] = parseFloat(window.prompt('Enter the monthly cost for '
                     + monthsArray[i], ''));

     document.write(monthsArray[i] + ' , ' + costsArray[i] + '<br/>');

     sum = sum + costsArray[i];
}

document.write('The monthly costs average is ' + getAverage(sum,
                costsArray.length));
```

لو افترضنا بأن الأرقام التي أدخلها المستخدم هي بالترتيب التالي:

111 112 113 311 256 654 223 123 545 234 233 111

مجموع هذه الأرقام هو 3026 ..

والنتيجة المكتوبة في المتصفح ستكون كالتالي:

```
January , 111
February , 112
March , 113
April , 311
May , 256
June , 654
July , 223
August , 123
September , 545
October , 234
November , 233
December , 111
The monthly costs average is 252.16666666666666
```

شكل المتوسِّط لم يعجبكم صح؟ :D

نستطيع هنا أن نستعمل الدّالة التي كتبناها سابقاً لتقريب الرقم إلى خانتين عشرية. الدّالة سنضعها في الأعلى مع الدالة الأولى ، وسنقوم باستدعائها هذه المرة في داخل الدّالة التي تقوم بحساب المتوسط:

```
function twoDPs(anyNumber)
{
     return Math.round(anyNumber * 100) / 100;
}
```

والآن نقوم بالتعديل على دالة حساب المتوسط ، وسنستدعي هذه الدالة في داخلها لتصبح بالشكل التالي:

```
function getAverage(total, number)
{
     return twoDPs(total / number);
}
```

الآن إذا افترضنا أن المستخدم أدخل الأرقام السابقة ، سيكون المجموع هو 3026 والعدد طبعاً 12

يعني ستكون دالة حساب المتوسط كالتالي:

```
function getAverage(3026, 12)
{
     return twoDPs(3026 / 12);
}
```

أول شيء ستتم عملية قسمة 3026 على 12 والناتج هو 252.16666666666666

ثم سينتقل هذا الناتج إلى دالّة تقريب الرقم إلى خانتين عشرية ، لتصبح دالة تقريب الأرقام بالشكل التالي:

```
function twoDPs(252.16666666666666)
{
     return Math.round(252.16666666666666 * 100) / 100;
}
```

وفي النهاية النتيجة التي سترجع من هذه الدالة هي 252.17 ، وسترجع هذه القيمة إلى دالة حساب المتوسط وليس إلى أمر الطباعة!

ودالّة حساب المتوسط ستقوم بدورها بإعادة هذه القيمة إلى المكان الذي استدعينا فيه الدالة.

الآن وبحمد الله اكتمل البرنامج معنا ، وهذا هو الكود النهائي:

```
function getAverage(total, number)
{
      return twoDPs(total / number);
}

function twoDPs(anyNumber)
{
      return Math.round(anyNumber * 100) / 100;
}

var costsArray = new Array(12);
var monthsArray = ['January','February','March','April','May','June','July'
                  ,'August','September','October','November','December'];
var sum = 0;

for ( var i = 0 ; i < costsArray.length ; i++ )
{
      costsArray[i] = parseFloat(window.prompt('Enter the monthly cost for '
                        + monthsArray[i], ''));

      document.write(monthsArray[i] + ' , ' + costsArray[i] + '<br/>');

      sum = sum + costsArray[i] ;
}
document.write('The monthly costs average is ' + getAverage(sum,
                  costsArray.length));
```

وإذا أدخلنا فيه نفس الأرقام السابقة ستكون النتيجة المكتوبة في المتصفح كالتالي:

```
January , 111
February , 112
March , 113
April , 311
May , 256
June , 654
July , 223
August , 123
September , 545
October , 234
November , 233
December , 111
The monthly costs average is 252.17
```

13- HTML Graphical User Interface :

سنتحدث هنا عن بعض أدوات واجهة المستخدم الخاصة بلغة HTML ، سأشرحها من ضمن الجافاسكربت لأنها لاتوجد لها فائدة إذا لم نستعملها مع أحد السكربتات كالجافاسكربت.

1- صناديق الكتابة Text boxes :

هي تحل محل النوافذ التي نظهرها للمستخدم لأخذ البيانات منه ، أي أنها تعتبر بديل لأمر window.prompt() ، وهي من وجهة نظري أكثر أناقة من النوافذ ، وتساعد على تنظيم وترتيب صفحة الانترنت أكثر من النوافذ التي تعلمنا عليها سابقاً.

2- الأزرار Buttons :

ووظيفتها هي تنفيذ أمر عند الضغط عليها من قبل المستخدم.

والأزرار وصناديق الكتابة يعتبرون مكملين لبعض ، لأنه غالباً سنقول للبرنامج أن يفعل شيء معين بالكتابة أو القيمة الموجودة في الـ text box عند الضغط على الزر.

الأزرار وصناديق الكتابة يجب أن تكون في داخل فورم ، ووسمه أو شفرته هي <form> </form> وهي أكواد HTML ، ولكل صندوق كتابة أو فورم اسم خاص به ، وسنرى فائدة كل هذا الآن.

لعمل صناديق الكتابة نستعمل الأكواد التالية:

```
<input type="text" name="anyName" value=""/>
```

هذا الكود لو وضعناه في صفحة HTML ، سيظهر لنا في الصفحة مكان أو صندوق للكتابة.

ولكن حتى الآن لاتوجد له فائدة لأننا لم نضع أوامر أو أكواد حتى تأخذ القيم المدخلة في هذا الصندوق. ونلاحظ بأنه يحتوي على ثلاثة عناصر هامة:

1- type="text" : يدل على أن النوع اللذي اخترناه هو "صندوق للكتابة" لأنه توجد أنواع أخرى مثل الأزرار وغيرها.

2- name="anyName" : هو اسم نضعه لهذا المربع وفائدة الاسم هي اذا أردنا أن نقوم بالتعامل مع هذا المربع عن طريق جافاسكربت سنذكر اسم المربع الذي نريد التعامل معه، ستتضح أكثر بعد قليل ان شاء الله.

3- value="" : هذا العنصر هو فائدة صندوق الكتابة الرئيسية ، غالباً سنترك قيمته فارغة ، والقيمة التي سيدخلها المستخدم في هذا الصندوق سيكون مكانها في هذا العنصر ، أي أننا إذا أردنا الحصول على ماقام المستخدم بكتابته في هذا الصندوق سنقوم بكتابة عنوان القيمة الموجودة هنا.

وفي النهاية قمنا بإغلاق هذا الكود بـ / ، لأنه ليس له شفرة إغلاق (مثل كود عرض الصور).

طيب دعونا ننتقل لطريقة عرض الأزرار الآن ، ثم سنشرح طريقة التعامل مع الاثنين.

لعرض زر في الصفحة ، سنستعمل الكود التالي:

```
<input type="button" value="Click here" onclick=""/>
```

1- type="button" : يدل على أن النوع اللذي اخترناه هو "زر أو button"

2- value="Click here" : هو الكلام الذي نريده أن يظهر على الزر ، مثل اضغط للتحويل أو اضغط هنا.

3- onclick="" : هذا هو أهم عنصر في الأزرار ، الأمر الذي سنكتبه هنا سيقوم بتنفيذه عندما يضغط المستخدم على الزر. وغالباً سيكون الأمر الموجود هنا عبارة عن استدعاء لدالّة.

ومثل سابقه .. في النهاية قمنا بإغلاق هذا الكود بـ / ، لأنه ليس له شفرة إغلاق (مثل كود عرض الصور).

إذا جمعنا الأمرين السابقين لعرض صندوق الكتابة والزر ، ووضعناهم بداخل فورم <form> ، وسمينا هذا الفورم بأي اسم ، سيكون الكود كامل كالتالي:

```
<form name="formName">
     <input type="text" name="anyName" value=""/>
     <input type="button" value="Click here" onclick=""/>
</form>
```

لو قمنا الآن بوضع هذا الكود في داخل ملف HTML بين شفرتي <body> </body> ، وفتحنا الملف بالمتصفِّح ، سنرى النتيجة التالية:

طبعاً شكل الأيقونة يختلف من نظام تشغيل لآخر ، فهذه الصورة مأخوذة من نظام تشغيل ماكنتوش وليس ويندوز ، فلا تستغربوا إذا ظهرت لديكم بشكل مختلف قليلاً.

الآن عرفنا كيف نعرضهم على الصفحة. دعونا ننتقل لطريقة التعامل معهم والاستفادة منهم عن طريق الجافاسكربت ..

البرنامج رقم 7 : التحويل من لتر إلى جالون

سنقوم بعمل برنامج يقوم بأخذ قيمة باللتر من المستخدم (عن طريق صناديق الكتابة) ومن ثم يحوِّلها إلى جالون ، مع العلم بأن الجالون يساوي 0.21998 لتر.

أول شيء سنقوم بعمل صندوقي كتابة ، **الأول** ليكتب فيه المستخدم كمية اللترات التي يريد تحويلها إلى جالون ، **والثاني** سنقوم بعرض نتيجة التحويل فيه (الكمية بالجالون). وطبعاً سنحتاج لعمل زر يقوم بعملية التحويل اذا ضغط عليه المستخدم.

فنكتب الأكواد الخاصة بعرض صندوقي الكتابة والزر كالتالي:

```
<form name="converter">
      Enter the value in litres:
      <br/>
      <input type="text" name="litres" value="" />
      <br/>
      <br/>
      <input type="button" value="Convert" onclick="" />
      <br/>
      <br/>
      The equivalent value in gallons is:
      <br/>
      <input type="text" name="gallons" value="" />
</form>
```

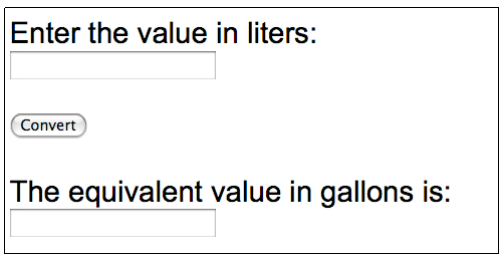
أولاً: كتبنا شفرة <form> وسميناه converter ، حتى نضع في داخله صندوقي الكتابة وزر التّحويل.

ثانياً: كتبنا أمر لعرض صندوق كتابة ، حتى يقوم المستخدم بكتابة الكمية باللتر في هذا الصندوق (ولذلك تركتنا قيمته value فارغة).

ثالثاً: كتبنا أمر لعرض زر ، حتى يقوم بعملية التحويل ، وتركنا العنصر الثالث فارغاً الآن وسنعود له بعد قليل.

رابعاً: كتبنا أمر لعرض صندوق كتابة ، نريد من البرنامج أن يقوم بعرض نتيجة التحويل فيه (ولذلك تركنا قيمته value فارغة).

إذا استعرضنا نتيجة هذا الكود في المتصفِّح ، ستكون كالتالي:



الآن كل ما قمنا بعمله هو مجرَّد منظر ، فهو لن يقوم بأي استجابة للمستخدم ، لأننا لم نضف له أي كود أو أمر جافاسكربت.

الآن سنعمل دالة تقوم بتحويل الكميات من لتر إلى جالون ، وسنسميها litresToGallons :

```
function litresToGallons(litres) {
     return litres * 0.21998;
}
```

إلى الآن لم نقم بشيء جديد ، فقط أنشأنا دالة تقوم بتحويل القيمة التي تأتي إليها من لتر إلى جالون.
القيمة التي أدخلها المستخدم في صندوق الكتابة الأول ، سنحصل عليها عن طريق هذا الأمر:

```
document.converter.litres.value
```

document : ثابت ولا يتغير.
converter : هو الاسم الذي أعطيناه للفورم الذي يحتوي على صندوق الكتابة، وضحت فائدة الاسم الآن؟
litres : هو اسم صندوق الكتابة الذي سنتعامل معه ، لأن مثل حالتنا يوجد صندوقي كتابة ، وعن طريق الاسم نستطيع التمييز بينهم.
value : هو المكان الذي كتب فيه المستخدم القيمة باللتر.

يعني صندوق الكتابة الثاني ، سيكون عنوان القيمة الخاصة فيه هو:

```
document.converter.gallons.value
```

الاختلاف الوحيد بين العنوانين هو اسم الصندوق فقط ، لأنهم موجودين في نفس الفورم ، وكلا الصندوقين ستكون قيمهم موجودة في value.

الآن سنقوم بعمل دالة ثانية ، وظيفتها هي عندما نستدعيها (بواسطة الزر) ستقوم بأخذ القيمة الموجودة في صندوق الكتابة الأول ، وسترسلها لدالة التحويل من لتر إلى جالون ، ثم ستقوم بإعادة النتيجة إلى صندوق الكتابة الثاني.

ببساطة أكثر ، نريد هذه الدّالة أن تقوم بتنفيذ هذا الأمر:

```
document.converter.gallons.value =
                litresToGallons(document.converter.litres.value)
```

يعني ستقوم بجعل قيمة صندوق الكتابة الثاني (الخاص بالنتيجة) يساوي: القيمة الموجودة في الصندوق الأول بعد تحويلها إلى لتر عن طريق دالّة التحويل.

بمعنى لو قام المستخدم بكتابة 20 في صندوق الكتابة الأول ، سيكون الأمر بالشكل التالي:

```
document.converter.gallons.value = litresToGallons(20)
```

بعدها سينتقل البرنامج إلى دالة التحويل من لتر إلى جالون ، بمعنى انه سيضرب رقم 20 في 0.21998 وستكون النتيجة 4.3996 ، وسيعود بهذه النتيجة وسيضعها في العنوان الخاص بصندوق الكتابة الثاني.

الآن سنكتب هذه الدالة:

```
function convert() {
      document.converter.gallons.value =
                        litresToGallons(document.converter.litres.value);
}
```

في هذه الدالّة لم نقم بوضع علامة أو اسم بين القوسين لأننا لانحتاج لذلك ، أو بمعنى آخر ، إذا استدعينا الدالّة عن طريق الزر لا نحتاج لإرسال قيمة لها.

الآن أصبح كل شيء جاهزاً للعمل ، تبقّى فقط أن نقوم باستدعاء الدالة الأخيرة بواسطة الزر ، فسنرجع لأكواد صناديق الكتابة والزر ، وسنضيف للزر استدعاء الدالة في عنصره الثالث onclick ، بالشكل التالي:

```
<form name="converter">
      Enter the value in litres:
      <br/>
      <input type="text" name="litres" value="" />
      <br/>
      <br/>
      <input type="button" value="Convert" onclick="convert()" />
      <br/>
      <br/>
      The equivalent value in gallons is:
      <br/>
      <input type="text" name="gallons" value="" />
</form>
```

هذه الأكواد ستكون بين شفرتي <body> </body>.

وأكواد الدّوال سنضعها في داخل شفرات <head> </head>.

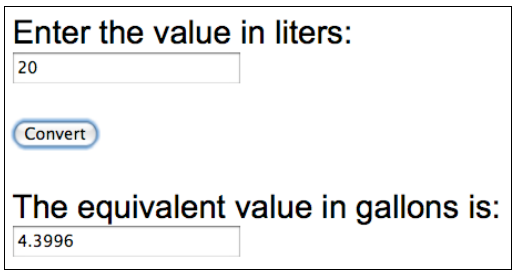
فإذا جمعنا كل ماقمنا به ، ووضعناه في ملف HTML كامل ، سيكون بالشكل التالي:

```
<html>
	<head>
		<title> Litres to Gallons Converter </title>

		function litresToGallons(litres)
		{
			return litres * 0.21998;
		}

		function convert()
		{
			document.converter.gallons.value =
					litresToGallons(document.converter.litres.value)
		}
	</head>
	<body>
		<form name="converter">
			Enter the value in litres:
			<br/>
			<input type="text" name="litres" value="" />
			<br/>
			<br/>
			<input type="button" value="Convert" onclick="convert()" />
			<br/>
			<br/>
			The equivalent value in gallons is:
			<br/>
			<input type="text" name="gallons" value="" />
		</form>
	</body>
</html>
```

فإذا كتبنا رقم 20 في المربع الأول ، ثم ضغطنا على زر التحويل ، ستظهر لنا النتيجة التالية:



طبعاً أهم شيء هنا ، هو أن نكون عرفنا كيف أخذنا القيمة التي كتبها المستخدم في مربع الكتابة ، وكيف جعلنا الزر يقوم بشيء معيَّن.

القيمة التي كتبها المستخدم في صندوق الكتابة ، أخذناها عن طريق هذا العنوان:

```
document.converter.litres.value
```

ثابت.اسم صندوق الكتابة.اسم الفورم.ثابت

وبالنسبة للزر ، كل ما قمنا بعمله هو استدعاء دالّة في عنصر onclick ، هذه الدّالة تقوم بأخذ القيمة من صندوق الكتابة الأول ، وتقوم بتحويلها أو التعديل عليها ، ومن ثم عرضها في صندوق الكتابة الثاني.

عنوان صندوق الكتابة الثاني هو:

```
document.converter.gallons.value
```

ثابت.اسم صندوق الكتابة.اسم الفورم.ثابت

طبعاً نستطيع تطوير البرنامج عن طريق إضافة دالّة تقريب الرقم إلى خانتين عشرية إذا أردنا ، عن طريق استدعائها في دالّة التحويل إلى جالون.

للتلخيص صناديق الكتابة والأزرار كلهم يحتاجون إلى ثلاث عناصر:

1- type : صناديق الكتابة نوعها text ، و الأزرار نوعها button .

2- صندوق الكتابة له اسم name ، و الأزرار لها value وتكون عبارة عن نص يظهر فوق الزر.

3- صندوق الكتابة له value وهو المكان الذي سيحمل القيمة الموجودة في الصندوق ، و الأزرار لها onclick ، وهي ستنفذ الأمر الموجود فيه عند الضغط على الزر.

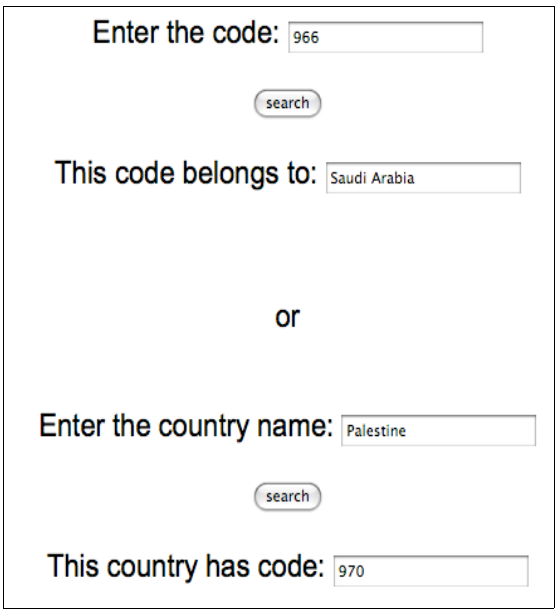
البرنامج رقم 8: البحث عن رمز الاتصال الدولي لدول العالم

فكرة البرنامج هي أن يستطيع المستخدم أن يبحث عن رمز الاتصال الدولي التابع لأي دولة ، عن طريق كتابة اسم الدولة الكامل ، أو البحث عن اسم الدولة الذي يتبع لها رمز اتصال دولي يدخله المستخدم. بمعنى انه يوجد قسمين:

1- أن يكتب اسم دولة ليحصل على رمز الهاتف الدولي الخاص بها.

2- أن يكتب رمز اتصال دولي ليعرف اسم الدولة التابعة له.

وهذه هي واجهة البرنامج:



في هذا البرنامج لن أشرح شيء ، ولكن سأكتب فكرة عمل البرنامج ، وللمعلومية ، نحن غير مطالبين بكتابة مثل هذا البرنامج ، فهو محاولة إجتهادية مني ، ولكن من يريد الاستفادة والتأكد من أنه فهم كل ماتم شرحه في الأقسام السابقة يجب أن يفهم طريقة عمل هذا البرنامج.

طريقة عمل البرنامج: في البداية أنشأت مصفوفتين:

1- الأولى: كتبت فيها أسماء الدول. **2- الثانية**: كتبت فيها رموز الاتصال الدولي لهذه الدول.

و يوجد هنا شرط ضروري ، وهو أن مراكز عناصر الدول في مصفوفة الدول ، هي نفس مراكز الرمز الدولي الخاص بها في مصفوفة رموز الاتصال. بمعنى لو افترضنا بأن مركز السعودية في مصفوفة الدول هو 22 يجب أن يكون مركز الرمز 966 هو 22 في مصفوفة رموز الاتصال الدولي وإلا لن يعمل البرنامج بشكل صحيح.

عندما يقوم المستخدم بكتابة اسم دولة ، فإن البرنامج سيبحث عن مركز هذه الدولة في مصفوفة الدول عن طريق مثود `indexOf` ، وبعدها سيقوم باستدعاء الرمز الموجود في نفس هذا المركز من مصفوفة رموز الاتصال الدولي.

ونفس الفكرة إذا قام المستخدم بالبحث عن رمز اتصال ، سيقوم البرنامج بتحديد مركز هذا الرمز في مصفوفة رموز الاتصال ، ثم سيقوم بعرض الدولة الموجودة في نفس هذا المركز في مصفوفة الدول.

طبعاً البرنامج يوجد فيه عيب، وهو أن المستخدم إذا أراد أن يبحث عن رمز الاتصال الدولي لدولة معينة مثل United States ، يجب أن يكتب نفس الاسم - وحجم الأحرف – بالضبط ، فلو كتب united states أو USA مثلاً ستكون النتيجة undefined لأن هذه الكلمات غير موجودة في المصفوفة.

أعتقد بأنه يمكن إصلاح مشكلة تطابق حجم الأحرف عن طريق مثود معيَّن يقوم بتحويل الأحرف إلى كبيرة أو صغيرة ، ولكنه غير مشروح في منهجنا ولذلك لم أبحث عنه. أو من الممكن وضع أسماء بديلة للدول التي يمكن أن تسمى بأكثر من اسم ولكنها تحتاج إلى مجهود ليس بالقليل.

البرنامج مرفق مع هذا الملف وكتبت فيه بعض التعليقات للتوضيح ، ومن أراد توضيح أكثر أو لديه استفسار أنا في الخدمة في أي وقت ، والبرنامج متاح للجميع سواء للاطلاع أو التعديل والتطوير أو غيره.

تمَّ بحمد الله

أتمنَّى أن أكون وفِّقت في شرح هذه اللغة الجميلة ببساطة وسهولة.
في حال وجود أخطاء إملائية أو قواعدية أو في الشرح أتمنى أن تعذروني.

أخوكم : **المعتصم**