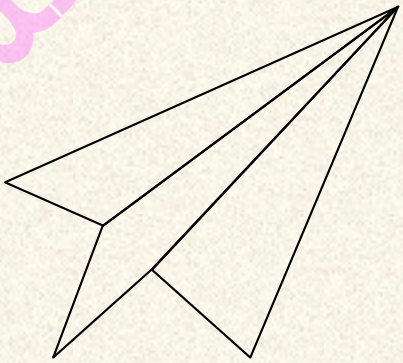
الكافى

فى

JavaScript

الجزء الثانى



ابو حبيب الحسينى

ملحوظة مهمة جدا

ان وجود الكلمات الانجليزية فى وسط الجمل العربية ينقل بعض الكلمات من مكانها فتظهر الجمل بشكل غير صحيح ويصعب فهما

مثال هذا الكلام

.(بدلاً من إلقاء خطأ)null أوundefined كان الكائنundefined يُرجع عامل التشغيل إذا.?

لاحظ هنا ان الجملة السابقة اصبحت غير مفهومة او غير مرتبة لان بعض الكلمات نقلت من مكانها بسبب وضع كلمات انجليزية وسط الجمل العربية فافى مثل هذة الحالات حاول ان تستنتج الجمله بنفسك وتفهما

حاولنا تقليل هذا العيب قدر المستطاع بستخدام بعض المصطلحات الانجليزية باللغة العربية مثل (سي اس اس) و (نود جى اس) وهكذا لنقلل من هذا قدر المستطاع

فهرس الكتاب

التحكم فى الوقت والتاريخ

تتيح لنا كائنات تاريخ جافاسكربت التعامل مع التواريخ:

أمثلة

```
const abibDate = new Date();
```



```
const abibDate = new Date("2022-03-25");
```


ملحوظة

". كائنات التاريخ ثابتة. الساعة ليست قيد التشغيل تستطيع ويتم تشغيلها بطرق كثيرة سنتطرق اليها ان شاء الله تعالى

التاريخ وطرق الإخراج

افتراضيًا، ستستخدم جافاسكربت المنطقة الزمنية للمتصفح يتم جلبها من الجهاز او الكمبيوتر او الموبايل وتعرض التاريخ كنصوص كاملة:

الأحد 19 نوفمبر 2023 الساعة 20:19:13 بتوقيت جرينتش+0200 (التوقيت الرسمي لأوروبا الشرقية)

. سوف تتعلم المزيد حول كيفية التحكم و العرض فى جميع اجزاء التواريخ، والوقت باذن الله تعالى في هذا الكتاب

طرق إنشاء كائنات التاريخ

يتم إنشاء كائنات التاريخ باستخدام المُنشئ new Date().

هناك 9 طرق لإنشاء كائن تاريخ جديد:

```
new Date()
new Date(date string)

new Date(year,month)
new Date(year,month,day)
new Date(year,month,day,hours)
new Date(year,month,day,hours,minutes)
new Date(year,month,day,hours,minutes,seconds)
new Date(year,month,day,hours,minutes,seconds,ms)

new Date(milliseconds)
```

Name	وصف
constructor	تُرجع الدالة التي أنشأت النموذج الأولي لكائن التاريخ
getDate()	إرجاع يوم الشهر (من 1 إلى 31)
getDay()	إرجاع يوم الأسبوع (من 0 إلى 6)
getFullYear()	يرجع السنة
getHours()	إرجاع الساعة (من 0 إلى 23)
getMilliseconds()	إرجاع المللي ثانية (من 0-999)
getMinutes()	إرجاع الدقائق (من 0-59)
getMonth()	إرجاع الشهر (من 0 إلى 11)
getSeconds()	إرجاع الثواني (من 0 إلى 59)
getTime()	إرجاع عدد المللي ثانية منذ منتصف ليل 1 يناير 1970 وتاريخ محدد
getTimezoneOffset()	إرجاع فارق التوقيت بين التوقيت العالمي المنسق (UTC) والتوقيت المحلي، بالدقائق
getUTCDate()	إرجاع يوم الشهر حسب التوقيت العالمي (من 1 إلى 31)
getUTCDay()	إرجاع يوم الأسبوع حسب التوقيت العالمي (من 0 إلى 6)
getUTCFullYear()	إرجاع السنة حسب التوقيت العالمي
getUTCHours()	إرجاع الساعة حسب التوقيت العالمي (من 0 إلى 23)
getUTCMilliseconds()	إرجاع المللي ثانية، وفقًا للتوقيت العالمي (من 0 إلى 999)

getUTCMinutes()	إرجاع الدقائق حسب التوقيت العالمي (من 0 إلى 59)
getUTCMonth()	إرجاع الشهر حسب التوقيت العالمي (من 0 إلى 11)
getUTCSeconds()	إرجاع الثواني حسب التوقيت العالمي (من 0 إلى 59)
getYear()	بدلاً من ذلك getFullYear() إهمال. استخدم طريقة
now()	إرجاع عدد الملي ثانية منذ منتصف ليل 1 يناير 1970
parse()	يوزع نص تاريخ ويعيد عدد الملي ثانية منذ 1 يناير 1970
prototype	يسمح لك بإضافة خصائص ودوال إلى كائن
setDate()	يضبط اليوم من الشهر لكائن التاريخ
setFullYear()	يضبط سنة كائن التاريخ
setHours()	يضبط ساعة كائن التاريخ
setMilliseconds()	يضبط الملي ثانية لكائن التاريخ
setMinutes()	قم بتعيين دقائق كائن التاريخ
setMonth()	يضبط شهر كائن التاريخ
setSeconds()	يضبط الثواني لكائن التاريخ
setTime()	يضبط تاريخًا لعدد محدد من الملي ثانية بعد/قبل 1 يناير 1970
setUTCDate()	يضبط اليوم من الشهر لكائن التاريخ، وفقًا للتوقيت العالمي
setUTCFullYear()	يضبط سنة كائن التاريخ، وفقًا للتوقيت العالمي
setUTCHours()	يضبط ساعة كائن التاريخ، وفقًا للتوقيت العالمي
setUTCMilliseconds()	يضبط الملي ثانية لكائن التاريخ، وفقًا للتوقيت العالمي
setUTCMinutes()	قم بتعيين دقائق كائن التاريخ، وفقًا للتوقيت العالمي
setUTCMonth()	يضبط شهر كائن التاريخ، وفقًا للتوقيت العالمي
setUTCSeconds()	قم بتعيين الثواني لكائن التاريخ، وفقًا للتوقيت العالمي
setYear()	بدلاً من ذلك setFullYear() إهمال. استخدم طريقة
toDateString()	يحول جزء التاريخ من كائن التاريخ إلى نص قابلة للقراءة
toGMTString()	بدلاً من ذلك toUTCString() إهمال. استخدم طريقة
toISOString()	إرجاع التاريخ كنص، باستخدام معيار ISO
toJSON()	إرجاع التاريخ كنص، بتنسيق تاريخ JSON
toLocaleDateString()	تقوم بإرجاع جزء التاريخ من كائن التاريخ كنص، باستخدام اصطلاحات اللغة
toLocaleTimeString()	إرجاع الجزء الزمني من كائن التاريخ كنص، باستخدام اصطلاحات

	اللغة
toLocaleString()	يحول كائن التاريخ إلى نص، باستخدام اصطلاحات اللغة
toString()	تحويل كائن التاريخ إلى نص
toTimeString()	يحول الجزء الزمني من كائن التاريخ إلى نص
toUTCString()	يحول كائن التاريخ إلى نص، وفقًا للتوقيت العالمي
UTC()	إرجاع عدد الملي ثانية في تاريخ ما منذ منتصف ليل 1 يناير 1970 UTC، وفقًا لتوقيت
valueOf()	إرجاع القيمة الأولية لكائن التاريخ

الاعلان عن كائن التاريخ

new Date() يقوم بإنشاء كائن تاريخ بالتاريخ والوقت الحاليين :

(مثال ✓ ⇩⇩)

```
const abibDate = new Date();
```

طريقة اخرى لكائن التاريخ (نص التاريخ)

new Date(date string) ينشئ كائن تاريخ من نص تاريخ :

أمثلة

```
const abibDate = new Date("October 13, 2014 11:13:00");
```



```
const abibDate = new Date("2022-03-25");
```


يتم وصف تنسيقات نص التاريخ في الفصل التالي.

وظيفة اخراج (السنة، الشهر)

. يقوم بإنشاء كائن تاريخ بتاريخ ووقت محددين new Date(year, month, ...)

أرقام تحدد السنة والشهر واليوم والساعة والدقيقة والثانية والميلي ثانية (بهذا الترتيب) 7:

(مثال ✓ ⇓⇓)

```
const abibDate = new Date(2018, 11, 24, 10, 33, 30, 0);
```


ملحوظة

جافاسكربت تحسب الأشهر من 0 إلى 11:

. يناير = 0

. ديسمبر = 11

لن يؤدي تحديد شهر أعلى من 11 إلى حدوث خطأ، ولكن سيتم إضافة الفائض إلى العام التالي:

تحديد:

```
const abibDate = new Date(2018, 15, 24, 10, 33, 30);
```


بالضبط مثل:

```
const abibDate = new Date(2019, 3, 24, 10, 33, 30);
```

لن يؤدي تحديد يوم أعلى من الحد الأقصى إلى حدوث خطأ، ولكن سيتم إضافة الفائض إلى الشهر التالي:

تحديد:

```
const abibDate = new Date(2018, 5, 35, 10, 33, 30);
```

بالضبط مثل:

```
const abibDate = new Date(2018, 6, 5, 10, 33, 30);
```

استخدام 6، 4، 3، أو 2 من الأرقام

أرقام تحدد السنة والشهر واليوم والساعة والدقيقة والثانية 6:

(مثال ✓ ⇓⇓)

```
const abibDate = new Date(2018, 11, 24, 10, 33, 30);
```


أرقام تحدد السنة والشهر واليوم والساعة والدقيقة 5:

(مثال ✓ ⇓⇓)

```
const abibDate = new Date(2018, 11, 24, 10, 33);
```


أرقام تحدد السنة والشهر واليوم والساعة 4:

(مثال ✓ ⇓⇓)

```
const abibDate = new Date(2018, 11, 24, 10);
```

3 أرقام تحدد السنة والشهر واليوم:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date(2018, 11, 24);
```


رقمان يحددان السنة والشهر:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date(2018, 11);
```


لا يمكنك حذف الشهر. إذا قمت بتوفير معلمة واحدة فقط، فسيتم التعامل معها على أنها ميلي ثانية.

(مثال ✓ ⇣⇣)

```
const abibDate = new Date(2018);
```

كيف الوصول الى القرن السابق

سيتم تفسير السنوات المكونة من رقم واحد أو رقمين على أنها 19xx:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date(99, 11, 24);
```

(✓ مثال ⇣⇣)

```
const abibDate = new Date(9, 11, 24);
```

تقوم بتخزين التواريخ بالمللي ثانية

تقوم جافاسكربت بتخزين التواريخ بعدد المللي ثانية منذ 01 يناير 1970.

الوقت صفر هو 01 يناير 1970 00:00:00 بالتوقيت العام .

يوم واحد (24 ساعة) هو 86400000 مللي ثانية.

الوقت الآن هو: 1700417953322 مللي ثانية بعد 01 يناير 1970

كيف الانشاء بـ (ملي ثانية)

ينشئ كائن تاريخ جديدًا بالمللي ثانية بالإضافة إلى وقت الصفر new Date(milliseconds):

أمثلة

01 يناير 1970 بالإضافة إلى 100000000000 مللي ثانية هو:

```
const abibDate = new Date(100000000000);
```


01 يناير 1970 ناقص 100000000000 مللي ثانية هو:

```
const abibDate = new Date(-100000000000);
```


01 ساعة هو 24 زائد 1970 يناير:

```
const abibDate = new Date(24 * 60 * 60 * 1000);
// or
const abibDate = new Date(86400000);
```


01 ملي ثانية هو 0 زائد 1970 يناير:

```
const abibDate = new Date(0);
```

طرق ووظائف التاريخ

عندما يتم إنشاء كائن التاريخ، هناك عدد من الطرق التي تسمح لك بالعمل عليه.

تسمح لك دوال التاريخ بالحصول على كائنات التاريخ للسنة والشهر واليوم والساعة والدقيقة والثانية والملي ثانية وتعيينها، باستخدام التوقيت المحلي أو التوقيت العام (UTC) (بتوقيت جرينتش أو العام).

عرض التواريخ

ستقوم جافاسكربت (افتراضيًا) بإخراج التواريخ بتنسيق نصوص كاملة:

(مثال ✓ ⇓⇓)

```
19 2023 20:19:13 GMT+0200 (Eastern European Standard Time)
```

toString(). عند عرض كائن تاريخ في ، يتم تحويله تلقائيًا إلى نص باستخدام الوظيفة

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.toString();
```


toDateString() تقوم الوظيفة بتحويل التاريخ إلى تنسيق أكثر قابلية للقراءة:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.toDateString();
```


toUTCString() تقوم الوظيفة بتحويل التاريخ إلى نص باستخدام معيار UTC:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.toUTCString();
```


toISOString() تقوم الوظيفة بتحويل التاريخ إلى نص باستخدام معيار ISO:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.toISOString();
```


استكمال مرجع التاريخ

للحصول على مرجع كامل للتاريخ، انتقل إلى:

يحتوي المرجع على أوصاف وأمثلة لجميع خصائص ودوال كل شى فى جافاسكربت.

تنسيقات التاريخ

طرق إدخال التاريخ

هناك بشكل عام ثلاثة أنواع من تنسيقات إدخال التاريخ في جافاسكربت:

يكتب	مثال
تاريخ الايزو	"2015-03-25" (The International Standard)
التاريخ القصير	"03/25/2015"
تاريخ طويل	"Mar 25 2015" or "25 Mar 2015"

معيارًا صارمًا في جافاسكربت ISO يتبع تنسيق.

التنسيقات الأخرى ليست محددة بشكل جيد وقد تكون خاصة بالمتصفح.

الإخراج الافتراضى للوقت والتاريخ

بغض النظر عن تنسيق الإدخال، ستقوم جافاسكربت (افتراضيًا) بإخراج التواريخ بتنسيق نصوص كاملة مثل هذا الاخراج:

Sun Nov 19 2023 20:19:41 GMT+0200 (Eastern European Standard Time)

تواريخ ISO

هو المعيار الدولي لتمثيل التواريخ والأوقات ISO 8601.

أيضًا تنسيق التاريخ فى جافاسكربت المفضل ISO 8601 (YYYY-MM-DD) يعد بناء جملة:

(التاريخ الكامل) (⇓⇓ مثال ✓)

```
const abibDate = new Date("2015-03-25");
```


سيكون التاريخ المحسوب مرتبطًا بمنطقتك الزمنية.
اعتمادًا على منطقتك الزمنية، فى الكمبيوتر او الجهاز اللوحى او الموبايل ستختلف النتيجة أعلاه بين 24 و25 مارس طبقا لنوع المتصفح .

ISO (السنة والشهر)

يمكن كتابة تواريخ ISO دون تحديد اليوم (YYYY-MM):

(⇓⇓ مثال ✓)

```
const abibDate = new Date("2015-03");
```


ستختلف المناطق الزمنية النتيجة المذكورة أعلاه بين 28 فبراير و01 مارس.

ISO (السنة فقط)

يمكن كتابة تواريخ ISO بدون الشهر واليوم (YYYY):

(⇓⇓ مثال ✓)

```
const abibDate = new Date("2015");
```


ستختلف المناطق الزمنية النتيجة المذكورة أعلاه بين 31 ديسمبر 2014 و01 يناير 2015.

ISO (التاريخ والوقت)

يمكن كتابة تواريخ ISO مع إضافة الساعات والدقائق والثواني (YYYY-MM-DDTHH:MM:SSZ):

(مثال ✔ ⇓⇓)

```
const abibDate = new Date("2015-03-25T12:00:00Z");
```


يتم فصل التاريخ والوقت بحرف T كبير.

يتم تعريف التوقيت العام المنسق (UTC) بالحرف الكبير Z.

بدلاً من ذلك HH:MM- أو HH:MM+ وأضف Z قم بإزالة، إذا كنت تريد تعديل الوقت بالنسبة إلى UTC:

(مثال ✔ ⇓⇓)

```
const abibDate = new Date("2015-03-25T12:00:00-06:30");
```


UTC (التوقيت العام المنسق) (توقيت جرينتش) هو نفس توقيت جرينتش.

المناطق الزمنية

عند تحديد تاريخ، دون تحديد المنطقة الزمنية، ستستخدم جافاسكربت المنطقة الزمنية للمتصفح او الكمبيوتر.

عند الحصول على تاريخ، دون تحديد المنطقة الزمنية، يتم تحويل النتيجة إلى المنطقة الزمنية للمتصفح او الكمبيوتر.

CDT (التوقيت) فسيتم تحويل التاريخ/الوقت إلى GMT (توقيت جرينتش)، بمعنى آخر: إذا تم إنشاء التاريخ/الوقت بتوقيت إذا كان المستخدم يتصفح من وسط الولايات المتحدة (الصيفي لوسط الولايات المتحدة.

كيف استخدام صيغ التواريخ المختصرة .

تتم كتابة التواريخ القصيرة باستخدام الصيغة "MM/DD/YYYY" كما يلي:

(⇓⇓ مثال ✓)

```
const abibDate = new Date("03/25/2015");
```

تحذيرات!

في بعض المتصفحات، قد تؤدي الأشهر أو الأيام التي لا تحتوي على أصفار بادئة إلى حدوث خطأ:

```
const abibDate = new Date("2015-3-25");
```

سلوك "YYYY/MM/DD" غير محدد.
ستحاول بعض المتصفحات القديمة تخمين التنسيق. سيعود بالتاريخ ناقص NaN.

```
const abibDate = new Date("2015/03/25");
```

سلوك "DD-MM-YYYY" غير محدد أيضًا.
ستحاول بعض المتصفحات القديمة تخمين التنسيق. سيعود البعض NaN.

```
const abibDate = new Date("25-03-2015");
```

استخدام صيغ التواريخ الطويلة.

كما يلي: "MMM DD YYYY" غالبًا ما تتم كتابة التواريخ الطويلة باستخدام هذه الصيغة

(⇓⇓ مثال ✓)

```
const abibDate = new Date("Mar 25 2015");
```

يمكن أن يكون الشهر واليوم بأي ترتيب:

(مثال ⇓⇓ ✓)

```
const abibDate = new Date("25 Mar 2015");
```


ويمكن كتابة الشهر كاملاً (يناير)، أو مختصراً :

(مثال ⇓⇓ ✓)

```
const abibDate = new Date("January 25 2015");
```


(مثال ⇓⇓ ✓)

```
const abibDate = new Date("Jan 25 2015");
```


يتم تجاهل الفواصل. الأسماء غير حساسة لحالة الأحرف:

(مثال ⇓⇓ ✓)

```
const abibDate = new Date("JANUARY, 25, 2015");
```

تحليل التواريخ

إذا كان لديك نص تاريخ صالحة، يمكنك استخدام الوظيفة Date.parse() لتحويلها إلى ميلي ثانية.

إرجاع عدد المللي ثانية بين التاريخ و1 يناير 1970 Date.parse():

(مثال ⇓⇓ ✓)

```
let msec = Date.parse("March 21, 2012");
```

يمكنك بعد ذلك استخدام عدد الملي ثانية لتحويله إلى كائن تاريخ:

(مثال ✓ ⇣⇣)

```
let msec = Date.parse("March 21, 2012");
const abibDate = new Date(msec);
```


استكمال مرجع التاريخ

للحصول على مرجع كامل للتاريخ، انتقل إلى:

المرجع الكامل لجافاسكربت

الحصول على طرق التاريخ

مُنشئ التاريخ

في جافاسكربت، يتم إنشاء كائنات التاريخ باستخدام new Date().

new Date() إرجاع كائن تاريخ بالتاريخ والوقت الحاليين.

الحصول على الوقت الحالي

```
const date = new Date();
```

طرق التاريخ

Method	وصف
getFullYear()	احصل على السنة كرقم مكون من أربعة أرقام (yyyy)
getMonth()	الحصول على الشهر كرقم (0-11)
getDate()	الحصول على اليوم كرقم (1-31)
getDay()	احصل على أيام الأسبوع كرقم (0-6)
getHours()	الحصول على ساعة (0-23)
getMinutes()	احصل على دقيقة (0-59)
getSeconds()	احصل على الثانية (0-59)
getMilliseconds()	الحصول على ميلي ثانية (0-999)
getTime()	الحصول على الوقت (مللي ثانية منذ 1 يناير 1970)

تُرجع طرق الحصول المذكورة أعلاه التوقيت المحلي .

في أسفل هذه الصفحة (UTC) تم توثيق التوقيت العام.

ملاحظة

تقوم دوال get بإرجاع المعلومات من كائنات التاريخ الموجودة.

"في كائن التاريخ، يكون الوقت ثابتًا. "الساعة" ليست "قيد التشغيل".

الوقت في كائن التاريخ ليس هو نفس الوقت الحالي.

وظيفة getFullYear()

تقوم الوظيفة getFullYear() بإرجاع سنة التاريخ كرقم مكون من أربعة أرقام:

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getFullYear();
```



```
const abibDate = new Date();
abibDate.getFullYear();
```


تحذير !

getYear(). قد يستخدم كود جافاسكربت القديم الوظيفة غير القياسية

بإرجاع سنة مكونة من رقمين getYear() من المفترض أن تقوم.

تم إهمالها getYear(). لا تستخدمها!

وظيفة getMonth()

بإرجاع شهر التاريخ كرقم (0-11) getMonth() تقوم الوظيفة.

ملحوظة1

...، نعيد للاهمية في جافاسكربت، يناير هو الشهر رقم 0، فبراير هو الشهر رقم 1

وأخيرا، ديسمبر هو الشهر رقم 11

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getMonth();
```



```
const abibDate = new Date();
abibDate.getMonth();
```


ملحوظة2

يمكنك استخدام مجموعة من الأسماء لإرجاع الشهر كاسم:

أمثلة

```
const months =
["January", "February", "March", "April", "May", "June", "July", "August", "September"
, "October", "November", "December"];

const abibDate = new Date("2021-03-25");
let month = months[abibDate.getMonth()];
```



```
const months =
["January", "February", "March", "April", "May", "June", "July", "August", "September"
, "October", "November", "December"];

const abibDate = new Date();
let month = months[abibDate.getMonth()];
```

وظيفة getDate()

تقوم الوظيفة getDate()(31-1) بإرجاع يوم التاريخ كرقم:

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getDate();
```



```
const abibDate = new Date();
abibDate.getDate();
```

وظيفة getHours()

تقوم الوظيفة getHours()(23-0) بإرجاع ساعات التاريخ كرقم:

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getHours();
```



```
const abibDate = new Date();
abibDate.getHours();
```

وظيفة getMinutes()

تقوم الوظيفة getMinutes()(59-0) بإرجاع دقائق التاريخ كرقم:

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getMinutes();
```



```
const abibDate = new Date();
abibDate.getMinutes();
```

وظيفة getSeconds ().

تقوم الوظيفة getSeconds()(59-0) بإرجاع ثواني التاريخ كرقم:

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getSeconds();
```



```
const abibDate = new Date();
abibDate.getSeconds();
```

وظيفة getMillithans ().

تقوم الوظيفة getMilliseconds()(999-0) بإرجاع الملي ثانية من التاريخ كرقم:

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getMilliseconds();
```



```
const abibDate = new Date();
abibDate.getMilliseconds();
```

وظيفة getDay()

تقوم الوظيفة getDay()(6-0) بإرجاع يوم الأسبوع من التاريخ كرقم.

ملحوظة1

في جافاسكربت، اليوم الأول من الأسبوع (اليوم 0) هو الأحد.

. بعض دول العام تعتبر أول يوم في الأسبوع هو يوم الاثنين والبعض يعتبره السبت تستطيع تغيره

أمثلة

```
const abibDate = new Date("2021-03-25");
abibDate.getDay();
```

```
const abibDate = new Date();
abibDate.getDay();
```


ملحوظة2

وإرجاع أيام الأسبوع كاسم getDay()، يمكنك استخدام مجموعة من الأسماء:

أمثلة

```
const days =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

const abibDate = new Date("2021-03-25");
let day = days[abibDate.getDay()];
```



```
const days =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

const abibDate = new Date();
let day = days[abibDate.getDay()];
```

وظيفة getTime()

تُرجع الوظيفة getTime() عدد الملي ثانية منذ 1 يناير 1970:

أمثلة

```
const abibDate = new Date("1970-01-01");
abibDate.getTime();
```

```
const abibDate = new Date("2021-03-25");
abibDate.getTime();
```



```
const abibDate = new Date();
abibDate.getTime();
```

وظيفة Date.now()

Date.now() تُرجع عدد الملي ثانية منذ 1 يناير 1970.

أمثلة

```
let ms = Date.now();
```


احسب عدد السنوات منذ 01/01/1970:

```
const minute = 1000 * 60;
const hour = minute * 60;
const day = hour * 24;
const year = day * 365;

let years = Math.round(Date.now() / year);
```


Date.now() هي وظيفة ثابتة لكائن التاريخ.

لا يمكنك استخدامه في كائن تاريخ مثل myDate.now().

بناء الجملة هو دائما Date.now().

طرق UTC

Method	Same As	وصف
getUTCDate()	getDate()	إرجاع تاريخ UTC
getUTCFullYear()	getFullYear()	إرجاع سنة UTC
getUTCMonth()	getMonth()	إرجاع شهر UTC
getUTCDay()	getDay()	إرجاع يوم UTC
getUTCHours()	getHours()	إرجاع ساعة UTC
getUTCMinutes()	getMinutes()	إرجاع دقائق UTC
getUTCSeconds()	getSeconds()	إرجاع ثواني UTC
getUTCMilliseconds()	getMilliseconds()	إرجاع ميلي ثانية UTC

UTC (التوقيت العام المنسق) تستخدم طرق الوقت العادية.

توقيت UTC هو نفس توقيت GMT (توقيت غرينتش).

يمكن أن يصل الفرق بين التوقيت المحلي والتوقيت العام المنسق (UTC) إلى 24 ساعة.

الوقت المحلي؟

التوقيت العام المنسق؟

وظيفة getTimezoneOffset ().

تقوم الوظيفة getTimezoneOffset() بإرجاع الفرق (بالدقائق) بين التوقيت المحلي وتوقيت UTC:

(⇓⇓ مثال ✓)

```
let diff = abibDate.getTimezoneOffset();
```

تعيين طرق التاريخ

إمكانية تعيين قيم التاريخ (السنوات، الأشهر، الأيام، الساعات، الدقائق، الثواني، الملي ثانية) لكائن التاريخ. تتيح لك دوال Set Date

---

ضبط طرق التاريخ

يتم استخدام دوال تعيين التاريخ لتعيين جزء من التاريخ:

Method	وصف
setDate()	تحديد اليوم كرقم (1-31)
setFullYear()	اضبط السنة (الشهر واليوم اختياريًا)
setHours()	ضبط الساعة (0-23)
setMilliseconds()	ضبط الملي ثانية (0-999)
setMinutes()	ضبط الدقائق (0-59)
setMonth()	ضبط الشهر (0-11)
setSeconds()	ضبط الثواني (0-59)
setTime()	ضبط الوقت (ملي ثانية منذ 1 يناير 1970)

وظيفة setFullYear()

تحدد الوظيفة setFullYear() 2020 حتى عام (✓ مثال ⇣⇣) سنة كائن التاريخ. في هذا ال:

(✓ مثال ⇣⇣)

```
const abibDate = new Date();
abibDate.setFullYear(2020);
```


يمكن للوظيفة تحديد الشهر واليوم اختياريًا setFullYear() :

(مثال ✔ ⇣⇣)

```
const abibDate = new Date();
abibDate.setFullYear(2020, 11, 3);
```

وظيفة setMonth ().

تحدد الوظيفة setMonth()(11-0) شهر كائن التاريخ:

(مثال ✔ ⇣⇣)

```
const abibDate = new Date();
abibDate.setMonth(11);
```


وظيفة setDate ().

تحدد الوظيفة setDate()(31-1) يوم كائن التاريخ:

(مثال ✔ ⇣⇣)

```
const abibDate = new Date();
abibDate.setDate(15);
```


يمكن أيضًا استخدام الوظيفة setDate() لإضافة أيام إلى تاريخ:

(مثال ✔ ⇣⇣)

```
const abibDate = new Date();
abibDate.setDate(abibDate.getDate() + 50);
```


إذا أردت إضافة الأيام إلى إزاحة الشهر أو السنة، فسيتم التعامل مع التغييرات تلقائيًا بواسطة كائن التاريخ.

وظيفة setHours()

تحدد الوظيفة setHours()(23-0) ساعات كائن التاريخ:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.setHours(22);
```


وظيفة setMinutes()

تقوم الوظيفة setMinutes()(59-0) بتعيين دقائق كائن التاريخ:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.setMinutes(30);
```


وظيفة setSeconds ().

تقوم الوظيفة setSeconds()(59-0) بتعيين الثواني لكائن التاريخ:

(مثال ✓ ⇣⇣)

```
const abibDate = new Date();
abibDate.setSeconds(30);
```

قارن التواريخ

يمكن بسهولة مقارنة التواريخ.

يقارن ال(✔ مثال ⇣⇣) التالي تاريخ اليوم بتاريخ 14 يناير 2100:

(✔ مثال ⇣⇣)

```
let abib = "";
const today = new Date();
const someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
  abib = "Today is before January 14, 2100.";
} else {
  abib = "Today is after January 14, 2100.";
}
```


نعيد للتذكير جافاسكربت تحسب الأشهر من 0 إلى 11. يناير هو 0. ديسمبر هو 11.

كائن الرياضيات

يتيح لك كائن Math إجراء مهام رياضية على الأرقام.

(✔ مثال ⇣⇣)

```
Math.PI;
```

الكلمةMath

Math على عكس الكائنات الأخرى، لا يحتوي كائن على مُنشئ.

Math كائن ثابت.

يمكن استخدام كافة الدوال والخصائص دون إنشاء كائن رياضي أولاً.

خصائص الرياضيات (الثوابت)

بناء جملة أي خاصية رياضية هو: Math.*property*.

توفر جافاسكربت 8 ثوابت رياضية يمكن الوصول إليها كخصائص رياضية:

(مثال ✓ ⬇⬇)

```
Math.E      // returns Euler's number
Math.PI     // returns PI
Math.SQRT2  // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2    // returns the natural logarithm of 2
Math.LN10   // returns the natural logarithm of 10
Math.LOG2E  // returns base 2 logarithm of E
Math.LOG10E  // returns base 10 logarithm of E
```

طرق الرياضيات

بناء جملة الرياضيات بأي وظيفة هو:Math.*method*(*number*)

هناك أربع طرق شائعة لتقريب رقم إلى عدد صحيح:

كيف تقريب الارقام

إرجاع أقرب عدد صحيحMath.round(x):

أمثلة

```
Math.round(4.6);
```



```
Math.round(4.5);
```



```
Math.round(4.4);
```


التقريب التصاعدى

Math.ceil(x) تُرجع قيمة x تقريب إلى أقرب عدد صحيح:

(مثال ✓ ⇓⇓)

```
Math.ceil(4.9);
Math.ceil(4.7);
Math.ceil(4.4);
Math.ceil(4.2);
Math.ceil(-4.2);
```

التقريب الى التنازلى

Math.floor(x) تُرجع قيمة x تقريب إلى أقرب عدد صحيح لها:

(مثال ✓ ⇓⇓)

```
Math.floor(4.9);
Math.floor(4.7);
Math.floor(4.4);
Math.floor(4.2);
Math.floor(-4.2);
```

كيف ازالة الكسور العشرية

Math.trunc(x) إرجاع الجزء الصحيح من **x:**

(مثال ✓ ⇣⇣)

```
Math.trunc(4.9);
Math.trunc(4.7);
Math.trunc(4.4);
Math.trunc(4.2);
Math.trunc(-4.2);
```

طرق الاختبار

Math.sign(x) تُرجع إذا كانت **x** موجبة أو فارغة أو سالبة:

(مثال ✓ ⇣⇣)

```
Math.sign(-4);
Math.sign(0);
Math.sign(4);
```

ايجاد القوة

Math.pow(x, y) إرجاع قيمة **x** إلى قوة **y:**

(✓ مثال ⇣⇣)

```
Math.pow(8, 2);
```

ايجاد الجزر التربيعى

Math.sqrt(x)لـ x: إرجاع الجذر التربيعي

(✓ مثال ⇣⇣)

```
Math.sqrt(64);
```

ايجاد القيمة المطلقة

Math.abs(x)لـ x: تُرجع القيمة المطلقة (الإيجابية)

(✓ مثال ⇣⇣)

```
Math.abs(-4.7);
```

طرق ايجاد زاوية الجيب

Math.sin(x) إرجاع جيب الزاوية (قيمة بين -1 و1) للزاوية x (المعطى بالراديان).

إذا كنت تريد استخدام الدرجات بدلاً من الراديان، فيجب عليك تحويل الدرجات إلى راديان:

الزاوية بالراديان = الزاوية بالدرجات x PI / 180.

(✓ مثال ⇣⇣)

```
Math.sin(90 * Math.PI / 180);     / returns 1 (the sine of 90 degrees)
```

تابع طرق زاوية الجيب

Math.cos(x) إرجاع جيب التمام (قيمة بين -1 و1) للزاوية x (المعطى بالراديان).

إذا كنت تريد استخدام الدرجات بدلاً من الراديان، فيجب عليك تحويل الدرجات إلى راديان:

الزاوية بالراديان = الزاوية بالدرجات x PI / 180.

(مثال ✓ ⇣⇣)

```
Math.cos(0 * Math.PI / 180);    // returns 1 (the cos of 0 degrees)
```

Math.max() و Math.min()

استخدامها للعثور على أدنى أو أعلى قيمة في قائمة الوسائط: Math.max() ويمكن Math.min()

(مثال ✓ ⇣⇣)

```
Math.min(0, 150, 30, 20, -8, -200);
```


(مثال ✓ ⇣⇣)

```
Math.max(0, 150, 30, 20, -8, -200);
```

توليد الأرقام العشوائية

Math.random() إرجاع رقم عشوائي بين 0 (شاملاً) و1 (حصرياً):

(⇣⇣ مثال ✓)

```
Math.random();
```


في الفصل التالي من هذا الكتاب Math.random() سوف تتعلم المزيد عنها.

وظيفة Math.log()

Math.log(x) إرجاع اللوغاريتم الطبيعي لـ x.

تُرجع اللوغاريتم الطبيعي الوقت اللازم للوصول إلى مستوى معين من الذيادة:

أمثلة

```
Math.log(1);
```



```
Math.log(2);
```



```
Math.log(3);
```


Math.log() و Math.E توأمان.

لنحصل على 10؟ Math.E كم مرة يجب أن نضرب

```
Math.log(10);
```

وظيفة Math.log2()

Math.log2(x) إرجاع اللوغاريتم الأساسي 2 لـ x.

كم مرة يجب أن نضرب 2 لنحصل على 8؟

```
Math.log2(8);
```

وظيفة Math.log10()

Math.log10(x) إرجاع اللوغاريتم الأساسي 10 لـ x.

كم مرة يجب أن نضرب 10 لنحصل على 1000؟

```
Math.log10(1000);
```

طرق الرياضيات

Method	وصف
abs(x)	إرجاع القيمة المطلقة لـ x
acos(x)	إرجاع قوس جيب التمام لـ x، بوحدات الراديان
acosh(x)	إرجاع قوس جيب التمام الزائدي لـ x
asin(x)	إرجاع قوس جيب x بالراديان
asinh(x)	إرجاع قوس الجيب الزائدي لـ x
atan(x)	إرجاع ظل الزاوية لـ x كقيمة رقمية بين -PI/2 وPI/2 راديان
atan2(y, x)	إرجاع قوس الظل لحاصل وسيطاته
atanh(x)	إرجاع قوس الظل الزائدي لـ x

cbrt(x)	إرجاع الجذر التكعيبي لـ x
ceil(x)	مع تقريبه لأعلى إلى أقرب عدد صحيح، x إرجاع
cos(x)	إرجاع جيب تمام x (بالراديان)
cosh(x)	إرجاع جيب التمام الزائدي لـ x
exp(x)	ترجع قيمة Ex
floor(x)	مع تقريبه للأسفل إلى أقرب عدد صحيح، x إرجاع
log(x)	إرجاع اللوغاريتم الطبيعي (الأساس E) لـ x
max(x, y, z, ..., n)	إرجاع الرقم ذو القيمة الأعلى
min(x, y, z, ..., n)	إرجاع الرقم ذو القيمة الأقل
pow(x, y)	إرجاع قيمة x إلى قوة y
random()	إرجاع رقم عشوائي بين 0 و 1
round(x)	تقريب x إلى أقرب عدد صحيح
sign(x)	تُرجع إذا كانت x (-1، 0، 1) سالبة أو فارغة أو موجبة
sin(x)	إرجاع جيب الزاوية لـ x (بالراديان)
sinh(x)	إرجاع جيب الزاوية الزائدي لـ x
sqrt(x)	إرجاع الجذر التربيعي لـ x
tan(x)	إرجاع ظل الزاوية
tanh(x)	إرجاع الظل الزائدي لرقم
trunc(x)	إرجاع الجزء الصحيح من الرقم (x)

الارقام العشوائية

كيف توليد رقم عشوائى

إرجاع رقم عشوائي بين 0 (شاملاً) و1 (حصرياً) Math.random()

(✓ مثال ⇓⇓)

```
// Returns a random number:
Math.random();
```


Math.random()1 تقوم دائمًا بإرجاع رقم أقل من.

الأعداد الصحيحة العشوائية

Math.random() المستخدمة مع Math.floor() يمكن استخدامها لإرجاع أعداد صحيحة عشوائية.

لا يوجد شيء اسمه أعداد صحيحة واعداد خاطئة في جافاسكربت.

نحن نتحدث هنا عن أرقام بدون كسور عشرية.

(✓ مثال ⇓⇓)

```
// Returns a random integer from 0 to 9:
Math.floor(Math.random() * 10);
```


(✓ مثال ⇓⇓)

```
// Returns a random integer from 0 to 10:
Math.floor(Math.random() * 11);
```


(✓ مثال ⇓⇓)

```
// Returns a random integer from 0 to 99:
Math.floor(Math.random() * 100);
```

(✓ مثال ⇣⇣)

```
// Returns a random integer from 0 to 100:
Math.floor(Math.random() * 101);
```


(✓ مثال ⇣⇣)

```
// Returns a random integer from 1 to 10:
Math.floor(Math.random() * 10) + 1;
```


(✓ مثال ⇣⇣)

```
// Returns a random integer from 1 to 100:
Math.floor(Math.random() * 100) + 1;
```

تابع توليد الارقام العشوائية

كما ترون من الأمثلة أعلاه، قد يكون من الجيد إنشاء دالة عشوائية مناسبة لاستخدامها في جميع أغراض الأعداد الصحيحة العشوائية.

تقوم دالة جافاسكربت هذه دائمًا بإرجاع رقم عشوائي بين الحد الأدنى (المضمن) والحد الأقصى (المستبعد):

(✓ مثال ⇣⇣)

```
function abibgetRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min) ) + min;
}
```


تقوم دالة جافاسكربت هذه دائمًا بإرجاع رقم عشوائي بين الحد الأدنى والحد الأقصى (كلاهما متضمن):

(✔ مثال ⇣⇣)

```
function abibgetRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1) ) + min;
}
```

الكلمات المنطقية

يمثل جافاسكربت Boolean إحدى القيمتين: true أو false .

القيم المنطقية

في كثير من الأحيان، في البرمجة، ستحتاج إلى نوع كلمات يمكن أن يحتوي على قيمة واحدة فقط من قيمتين، مثل

- نعم / لا
- تشغيل/إيقاف
- خطأ صحيح

. ولهذا السبب، تحتوي جافاسكربت على نوع كلمات منطقي . يمكن أن تأخذ القيم صحيحة أو خاطئة فقط

الدالة بوليين

يمكنك استخدام Boolean() الدالة لمعرفة ما إذا كان البرامتر (أو المتغير) صحيحًا:

(✔ مثال ⇣⇣)

```
Boolean(10 > 9)
```


أو حتى أسهل:

(✔ مثال ⇣⇣)

```
(10 > 9)
10 > 9
```

المقارنات والشروط

نظرة عامة كاملة على عوامل المقارنة **JS Comparisons** يقدم الفصل.

نظرة عامة كاملة على الكلمات الشرطية **JS** يقدم الفصل شروط.

وهنا بعض الأمثلة:

Operator	وصف	مثال
==	يساوي	if (day == "Monday")
>	أكثر من	if (salary > 9000)
<	أقل من	if (age < 18)

القيمة المنطقية للتعبير هي الأساس لجميع مقارنات وشروط جافاسكربت.

كل شيء له "قيمة" يعتبر صحيح

أمثلة

- 100
- 3.14
- -15
- "Easy-to"
- "false"
- 7 + 1 + 3.14

كتاب الكافى فى جافاسكربت الجزء الثانى ابو حبيب الحسينى

كل شيء بدون "قيمة" يعتبر خطا

القيمة المنطقية 0 (صفر) خاطئة :

```
let x = 0;
Boolean(x);
```


القيمة المنطقية لـ -0 (ناقص الصفر) خاطئة :

```
let x = -0;
Boolean(x);
```


القيمة المنطقية لـ "" (نص فارغة) غير صحيحة :

```
let x = "";
Boolean(x);
```


القيمة المنطقية غير المحددة غير صحيحة :

```
let x;
Boolean(x);
```


القيمة المنطقية للقيمة الخالية خاطئة :

```
let x = null;
Boolean(x);
```

القيمة المنطقية للخطأ هي false :

```
let x = false;
Boolean(x);
```


القيمة المنطقية لـ NaN خاطئة :

```
let x = 10 / "Hallo";
Boolean(x);
```

تابع Booleans

عادةً ما تكون القيم المنطقية لجافاسكربت عبارة عن قيم بدائية تم إنشاؤها من القيم الحرفية:

```
let x = false;
```

ولكن يمكن أيضًا تعريف القيم المنطقية على أنها كائنات بالكلمة المحجوزة new:

```
let y = new Boolean(false);
```

(مثال ⇓⇓ ✓)

```
let x = false;
let y = new Boolean(false);

// typeof x returns boolean
// typeof y returns object
```

لا تقم بإنشاء كائنات منطقية.

تعمل الكلمة new على تعقيد ال اكواد وإبطاء سرعة التنفيذ المحجوزة.

يمكن أن تنتج الكائنات المنطقية نتائج غير متوقعة:

عند استخدام المعامل == ، x و y متساويان :

```
let x = false;
let y = new Boolean(false);
```


عند استخدام المعامل === ، x و y غير متساويين :

```
let x = false;
let y = new Boolean(false);
```


لاحظ الفرق بين (x==y) و(x===y).

صحيحة أو خاطئة؟(x == y)

```
let x = new Boolean(false);
let y = new Boolean(false);
```


صحيحة أو خاطئة؟(x === y)

```
let x = new Boolean(false);
let y = new Boolean(false);
```


دائمًا ما تؤدي المقارنة بين كائنين في جافاسكربت إلى ظهور خطأ.

المقارنات

يتم استخدام عوامل المقارنة والمنطقية لاختبار **false** أو **true**.

عوامل المقارنة

تُستخدم عوامل المقارنة في الكلمات المنطقية لتحديد المساواة أو الاختلاف بين المتغيرات أو القيم.

يوضح الجدول أدناه عوامل المقارنة، بالنظر إلى ذلك **x = 5**:

Operator	وصف	Comparing	Returns
==	يساوي	x == 8	false
		x == 5	true
		x == "5"	true
===	قيمة متساوية ونوع متساو	x === 5	true
		x === "5"	false
!=	غير متساوي	x != 8	true
!==	قيمة غير متساوية أو نوع غير متساوٍ	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	أكثر من	x > 8	false
<	أقل من	x < 8	true
>=	أكبر من أو يساوي	x >= 8	false
<=	اقل او يساوي	x <= 8	true

كيف يمكن استخدام عوامل المقارنة

يمكن استخدام عوامل المقارنة في العبارات الشرطية لمقارنة القيم واتخاذ الإجراءات بناءً على النتيجة:

```
if (age < 18) abib = "Too young to buy alcohol";
```

سوف تتعلم المزيد عن استخدام العبارات الشرطية في الفصل التالي من هذا الكتاب.

المعاملات المنطقية

يتم استخدام العوامل المنطقية لتحديد المنطق بين المتغيرات أو القيم.

يشرح الجدول أدناه العوامل المنطقية، y = 3 وx = 6 بالنظر إلى ذلك:

Operator	Description	مثال
&&	و	(x < 10 && y > 1) is true
||	او	(x == 5 || y == 5) is false
!	ليس	!(x == y) is true

العامل الشرطي (الثلاثي).

تحتوي جافاسكربت أيضًا على عامل شرطي يعين قيمة لمتغير بناءً على شرط ما.

بناء الجملة

```
variableName = (condition) ? value1:value2
```

(مثال ✓ ⇣⇣)

```
let Habib= (age < 18) ? "Too young":"Old enough";
```


إذا كان متغير العمر قيمة أقل من 18 عامًا، فإن قيمة المتغير القابل للتصويت ستكون "صغير جدًا"، وإلا فإن قيمة القابل للتصويت ستكون "كبير بما يكفي".

مقارنة الأنواع المختلفة

```
age = Number(age);
if (isNaN(age)) {
  Habib= "Input is not a number";
} else {
  Habib= (age < 18) ? "Too young" : "Old enough";
}
```

كيف استخدام عامل الدمج الفارغ (؟؟)

التشغيل بإرجاع الوسيطة الأولى إذا لم تكن فارغة ?? يقوم عامل (null أو undefined).

وإلا فإنه يقوم بإرجاع الوسيطة الثانية.

(مثال ✔ ⇣⇣)

```
let Name = null;
let abib = "missing";
let result = Name ?? Habib;
```


مدعوم في جميع المتصفحات منذ مارس 2020 nullish عامل التشغيل:

مشغل التسلسل الاختياري (؟.)

يُرجع عامل التشغيل إذا؟. undefined كان الكائن undefined أو null (بدلاً من إلقاء خطأ).

(مثال ✔ ⇣⇣)

```
// Create an object:
const $Habib= {use:"Mohamed", age:"20", color:"white"};
// Ask for $Name:
document.getElementById("Habib").innerHTML = $Habib?.Name;
```

مشغل التسلسل الاختياري مدعوم في جميع المتصفحات منذ مارس 2020:

كيف استخدام صيغة إذا، وإلا، وإلا إذا فى البرمجة

تُستخدم العبارات الشرطية لتنفيذ إجراءات مختلفة بناءً على شروط مختلفة.

عبارات شرطية

في كثير من الأحيان، عند كتابة ال اكواد ، تريد تنفيذ إجراءات مختلفة لقرارات مختلفة.

يمكنك استخدام العبارات الشرطية في ال اكواد للقيام بذلك.

في جافاسكربت لدينا العبارات الشرطية التالية:

- لتحديد كتلة من ال اكواد المراد تنفيذها، إذا كان الشرط المحدد صحيحًا **if** يُستخدم
- لتحديد كتلة من ال اكواد المراد تنفيذها، إذا كان نفس الشرط خاطئًا **else** يُستخدم
- لتحديد شرط جديد للاختبار، إذا كان الشرط الأول خاطئًا **else if** يُستخدم
- لتحديد العديد من كتل ال اكواد البديلة التي سيتم تنفيذها **switch** يُستخدم

ويرد كلمة الكلمة في الفصل التالي **switch**.

كيف وضع شرط فى الكود

استخدم **if** العبارة لتحديد كتلة من تعليمات جافاسكربت البرمجية ليتم تنفيذها إذا كان الشرط صحيحًا.

بناء الجملة

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

إلى إنشاء خطأ جافاسكربت (IF أو IF) بالأحرف الصغيرة. ستؤدي الأحرف الكبيرة if لاحظ أنه.

(مثال ✔ ⇣⇣)

قم بتحية "يوم جيد" إذا كانت الساعة أقل من 18:00:

```
if (hour < 18) {
  $Habib = "Good day";
}
```

ستكون نتيجة التحية:

```
//Good day
```

اذا الشرط صحيح افعل كذا والا افعل كذا

استخدم else العبارة لتحديد كتلة من ال اكواد ليتم تنفيذها إذا كان الشرط وعبارة اخرى يتم تنفيذه اذا كان غير صحيح.

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

(مثال ✔ ⇣⇣)

إذا كانت الساعة أقل من 18، قم بإنشاء تحية "يوم جيد"، وإلا "مساء الخير":

```
if (hour < 18) {
  $Habib = "Good day";
} else {
  $Habib = "Good evening";
}
```

ستكون نتيجة التحية:

```
//Good evening
```

كلمة والا إذا

استخدم else if العبارة لتحديد شرط جديد إذا كان الشرط الأول خطأ.

بناء الجملة

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

(⇓⇓ مثال ✓)

إذا كان الوقت أقل من 10:00، فقم بإنشاء تحية "صباح الخير"، وإذا لم يكن الأمر كذلك، ولكن الوقت أقل من 20:00، فقم بإنشاء تحية "يوم جيد"، وإلا "مساء الخير":

```
if (time < 10) {
  $Habib = "Good morning";
} else if (time < 20) {
  $Habib = "Good day";
} else {
  $Habib = "Good evening";
}
```

صيغة اخرى لوضع الشروط

لتنفيذ إجراءات مختلفة بناءً على شروط مختلفة switch يتم استخدام العبارة.

استخدام الكلمة سيوتش

العبارة لتحديد إحدى كتل ال اكواد العديدة التي سيتم تنفيذها switch استخدم.

بناء الجملة

```
switch(expression) {
 case x:
  // code block
  break;
 case y:
  // code block
  break;
 default:
```

```
  // code block
}
```

هذه هي الوظيفة التي يعمل بها:

- يتم تقييم تعبير التبديل مرة واحدة.
- وتتم مقارنة قيمة البرامتر مع قيم كل حالة.
- إذا كان هناك تطابق، فسيتم تنفيذ كتلة ال اكواد المرتبطة.
- إذا لم يكن هناك تطابق، فسيتم تنفيذ كتلة ال اكواد الافتراضية.

(⇣⇣ مثال ✓)

تقوم الوظيفة **getDay()** بإرجاع أيام الأسبوع كرقم بين **0** و**6**.

(الأحد=0، الاثنين=1، الثلاثاء=2..)

يستخدم هذا ال(✓ مثال ⇣⇣) رقم يوم الأسبوع لحساب اسم يوم الأسبوع:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
```

```
    day = "Saturday";
}
```

الكلمة بريك

break عندما تصل جافاسكربت إلى كتلة التبديل ، فإنها تخرج من كتلة التبديل كلمة محجوزة.

سيؤدي هذا إلى إيقاف التنفيذ داخل كتلة التبديل.

ليس من الضروري كسر الحالة الأخيرة في كتلة التبديل. تنكسر الكتلة (تنتهي) هناك على أي حال.

ملاحظة: إذا قمت بحذف عبارة Break، فسيتم تنفيذ الحالة التالية حتى لو لم يتطابق التقييم مع الحالة.

وضع الشرط الافتراضى

default تحدد الكلمة المحجوزة الكود الذي سيتم تشغيله في حالة عدم وجود تطابق لحالة الأحرف:

(مثال ✓ ⇣⇣)

getDay() تقوم الوظيفة بإرجاع أيام الأسبوع كرقم بين 0 و6.

إذا كان اليوم ليس السبت (6) ولا الأحد (0)، فاكتب رسالة افتراضية:

```
switch (new Date().getDay()) {
  case 6:
    abib = "Today is Saturday";
    break;
  case 0:
    abib = "Today is Sunday";
    break;
  default:
```

```
    abib = "Looking forward to the Weekend";
  }
```

نتيجة النص ستكون:

Today is Sunday

هي الحالة الأخيرة في كتلة التبديل **default** لا يجب أن تكون الحالة:

(⇓⇓ مثال ✔)

```
switch (new Date().getDay()) {
  default:
    abib = "Looking forward to the Weekend";
    break;
  case 6:
    abib = "Today is Saturday";
    break;
  case 0:
    abib = "Today is Sunday";
}
```


إذا **default** لم تكن الحالة الأخيرة في كتلة التبديل، فتذكر إنهاء الحالة الافتراضية بفاصل.

كتل ال اكواد المشتركة

في بعض الأحيان قد ترغب في استخدام حالات تبديل مختلفة لنفس الرمز.

في هذا ال(✔ مثال ⇓⇓)، تشترك الحالة **4** و**5** في نفس كتلة ال اكواد ، وتشترك **0** و**6** في كتلة تعليمات برمجية أخرى:

(✓ مثال ⇣⇣)

```
switch (new Date().getDay()) {
  case 4:
  case 5:
    abib = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    abib = "It is Weekend";
    break;
  default:
    abib = "Looking forward to the Weekend";
}
```

تفاصيل التبديل

. إذا تطابقت حالات متعددة مع قيمة حالة، فسيتم تحديد الحالة الأولى

. إذا لم يتم العثور على أي حالات مطابقة، فسيستمر البرنامج في التسمية الافتراضية

. إذا لم يتم العثور على تسمية افتراضية، فسيستمر البرنامج في العبارة (العبارات) بعد التبديل

المقارنة الصارمة

تستخدم حالات التبديل مقارنة صارمة (===).

يجب أن تكون القيم من نفس النوع للمطابقة.

لا يمكن أن تكون المقارنة الصارمة صحيحة إلا إذا كانت المعاملات من نفس النوع.

في هذا ال x: لن يكون هناك تطابق لـ (⇣⇣ مثال ✓)

(✓ مثال ⇣⇣)

```
let x = "0";
switch (x) {
  case 0:
    abib = "Off";
    break;
```

```
  case 1:
    abib = "On";
    break;
  default:
    abib = "No value found";
}
```

الحلقة

يمكن للحلقات تنفيذ كتلة من ال اكواد عدة مرات.

مثال

تعتبر الحلقات مفيدة إذا كنت تريد تشغيل نفس الكود مرارًا وتكرارًا، وفي كل مرة بقيمة مختلفة.

غالبًا ما يكون هذا هو الحال عند العمل مع المصفوفات:

بدلاً من الكتابة:

```
abib += HaBiB[0] + "<br>";
abib += HaBiB[1] + "<br>";
abib += HaBiB[2] + "<br>";
abib += HaBiB[3] + "<br>";
abib += HaBiB[4] + "<br>";
abib += HaBiB[5] + "<br>";
```

يمكنك كتابة:

```
for (let i = 0; i < HaBiB.length; i++) {
  abib += HaBiB[i] + "<br>";
}
```

أنواع مختلفة من الحلقات

تدعم جافاسكربت أنواعًا مختلفة من الحلقات:

- for- حلقات من خلال كتلة من ال اكواد عدة مرات
- for/in- حلقات من خلال خصائص الكائن
- for/of- حلقات من خلال قيم كائن قابل للتكرار
- while- حلقات من خلال كتلة من ال اكواد عندما يكون الشرط المحدد صحيحا
- do/while- يتكرر أيضًا خلال كتلة من ال اكواد عندما يكون الشرط المحدد صحيحًا

For الحلقة

حلقة تحتوي على 3 تعبيرات اختيارية for تنشئ العبارة:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

- يتم تنفيذ البرامتر 1 (مرة واحدة) قبل تنفيذ كتلة ال اكواد
- يحدد البرامتر 2 شرط تنفيذ كتلة ال اكواد
- يتم تنفيذ البرامتر 3 (في كل مرة) بعد تنفيذ مقطع ال اكواد

(✓ مثال ⇣⇣)

```
for (let i = 0; i < 5; i++) {
  abib += "The number is " + i + "<br>";
}
```


ومن ال(✓ مثال ⇣⇣) أعلاه يمكنك أن تقرأ:

يقوم البرامتر 1 بتعيين متغير قبل بدء الحلقة (دع i = 0).

يحدد البرامتر 2 شرط تشغيل الحلقة (أقل من i 5 يجب أن يكون).

يزيد البرامتر 3 القيمة (i++) في كل مرة يتم فيها تنفيذ كتلة ال اكواد في الحلقة.

البرامتر 1

عادةً ستستخدم البرامتر 1 لتهيئة المتغير المستخدم في الحلقة

هذا ليس هو الحال دائما. جافاسكربت لا يهتم. البرامتر 1 اختياري.

يمكنك بدء العديد من القيم في البرامتر 1 (مفصولة بفاصلة):

(مثال ✓ ⇣⇣)

```
for (let i = 0, len = HaBiB.length, abib = ""; i < len; i++) {
  abib += HaBiB[i] + "<br>";
}
```


ويمكنك حذف البرامتر 1 (مثلما يتم تعيين القيم قبل بدء الحلقة):

(مثال ✓ ⇣⇣)

```
let i = 2;
let len = HaBiB.length;
let abib = "";
for (; i < len; i++) {
  abib += HaBiB[i] + "<br>";
}
```


البرامتر 2

غالبًا ما يتم استخدام البرامتر 2 لتقييم حالة المتغير الأولي.

هذا ليس هو الحال دائما. جافاسكربت لا يهتم. البرامتر 2 اختياري أيضًا.

إذا عاد البرامتر 2 إلى القيمة الصحيحة، فستبدأ الحلقة من جديد. إذا عادت خطأ، ستنتهي الحلقة.

إذا قمت بحذف البرامتر 2، فيجب عليك توفير فاصل داخل الحلقة. وإلا فإن الحلقة لن تنتهي أبدا. سيؤدي هذا إلى تعطل المتصفح. اقرأ عن فترات الايقاف في فصل لاحق من هذا الكتاب.

البرامتر 3

غالبًا ما يزيد البرامتر 3 قيمة المتغير الأولي.

هذا ليس هو الحال دائما. جافاسكربت لا يهتم. البرامتر 3 اختياري.

أو أي شيء آخر. (i = i + 15) أو الزيادة الموجبة. (i--) يمكن للتعبير 3 أن يفعل أي شيء مثل الزيادة السالبة.

يمكن أيضًا حذف البرامتر 3 (مثلًا عند زيادة قيمك داخل الحلقة):

(مثال ✓ ⇣⇣)

```
let i = 0;
let len = HaBiB.length;
let abib = "";
for (; i < len; ) {
  abib += HaBiB[i] + "<br>";
  i++;
}
```

نطاق الحلقة

في حلقة var باستخدام:

(مثال ✓ ⇣⇣)

```
var i = 5;

for (var i = 0; i < 10; i++) {
  // some code
}

// Here i is 10
```


في حلقة let باستخدام:

(✓ مثال ⇣⇣)

```
let i = 5;

for (let i = 0; i < 10; i++) {
  // some code
}

// Here i is 5
```


يقوم المتغير المعلن في الحلقة بإعادة تعريف المتغير الموجود خارج الحلقة، var في ال(✓ مثال ⇣⇣) الأول، باستخدام

لا يقوم المتغير المُعلن في الحلقة بإعادة تعريف المتغير خارج الحلقة، let في ال(✓ مثال ⇣⇣) الثاني، باستخدام

مرئيًا فقط داخل الحلقة i في حلقة، سيكون المتغير i استخدامه للإعلان عن المتغير let عند

في الفصل التالي for/of والحلقة for/in سيتم شرح الحلقة.

حلقة بينما

في الفصول التالية the do/while و while سيتم شرح الحلقة.

حلقة For In

عبر خصائص الكائن for in يتم تنفيذ عبارة جافاسكربت:

بناء الجملة

```
for (key in object) {
  // code block to be executed
}
```

(✓ مثال ⇣⇣)

```
const hAbiB = {fName:"Habib", lName:"Al Husini 😀😁😂", age:18};

let abib = "";
```

```
for (let x in hAbiB) {
  abib += hAbiB[x];
}
```


مثال اخر

- for in تتكرر حلقة على كائن عنصر
- (x) كل تكرار يقوم بإرجاع مفتاح
- يتم استخدام المفتاح للوصول إلى قيمة المفتاح
- [x]قيمة المفتاح هي العنصر

التنفيذ على المصفوفة

for inفوق خصائص المصفوفة يمكن أيضًا تنفيذ عبارة جافاسكربت:

بناء الجملة

```
for (variable in array) {
  code
}
```

(مثال ✓ ⇣⇣)

```
const abibNum = [45, 4, 9, 16, 25];

let hAbiB = "";
for (let x in abibNum) {
  hAbiB += abibNum[x];
}
```


for in over a Array لا تستخدم إذا كان ترتيب الفهرس مهمًا.

يعتمد ترتيب الفهرس على التنفيذ، وقد لا يمكن الوصول إلى قيم ال مصفوفة بالترتيب الذي تتوقعه.

عندما يكون الترتيب مهمًا Array.forEach() أو ، for أو حلقة ، for من الأفضل استخدام حلقة.

---

forEach () على المصفوفة

forEach() تستدعي الوظيفة دالة (دالة رد اتصال) مرة واحدة لكل عنصر من عناصر المصفوفة.

(مثال ✓ ⇣⇣)

```
const abibNum = [45, 4, 9, 16, 25,45, 4, 9, 16, 25];

let hAbiB = "";
abibNum.forEach(Husini);

function Husini(value, index, array) {
  hAbiB += value;
}
```


لاحظ أن الدالة تأخذ 3 وسيطات:

- قيمة العنصر
- فهرس البند
- المصفوفة نفسها

يستخدم ال(مثال ✓ ⇣⇣) أعلاه معلمة القيمة فقط. يمكن إعادة كتابته إلى:

(مثال ✓ ⇣⇣)

```
const abibNum = [45, 4, 9, 16, 25];

let hAbiB = "";
abibNum.forEach(Husini);

function Husini(value) {
  hAbiB += value;
}
```

تابع انواع الحلقات

for of يتم تنفيذ عبارة جافاسكربت عبر قيم كائن قابل للتكرار.

فهو يتيح لك التكرار عبر هياكل الكلمات القابلة للتكرار مثل المصفوفات والنصوص والخرائط وقوائم العقد والمزيد:

بناء الجملة

```
for (variable of iterable) {
  // code block to be executed
}
```

متغير - لكل تكرار يتم تعيين قيمة الخاصية التالية للمتغير. يمكن الإعلان عن المتغير بـ const أو let أو var.

قابل للتكرار - كائن له خصائص قابلة للتكرار.

التكرار فوق مصفوفة

(مثال ⇣⇣ ✓)

```
const HaBiB= ["Abu Habib Al-Husini", "Hamza", "Mini"];

let abib = "";
for (let x of HaBiB) {
  abib += x;
}
```

التكرار على نص

(مثال ⇣⇣ ✓)

```
let language = "javascript";

let abib = "";
for (let x of language) {
abib += x;
}
```

في الفصل التالي do/while والحلقة while سيتم شرح الحلقة.

أثناء الحلقة

يمكن للحلقات تنفيذ كتلة من ال اكواد طالما كان الشرط المحدد صحيحًا.

حلقة "بينما".

تتكرر الحلقة while عبر كتلة من ال اكواد طالما كان الشرط المحدد صحيحًا.

بناء الجملة

```
while (condition) {
  // code block to be executed
}
```

(✓ مثال ⇣⇣)

أقل من 10 (i) في ال(✓ مثال ⇣⇣) التالي، سيتم تشغيل ال اكواد الموجودة في الحلقة مرارًا وتكرارًا، طالما أن المتغير:

(✓ مثال ⇣⇣)

```
while (i < 10) {
  abib += "The number is " + i;
  i++;
}
```


إذا نسيت زيادة المتغير المستخدم في الشرط، فلن تنتهي الحلقة أبدًا. سيؤدي هذا إلى تعطل المتصفح.

حلقة "افعل أثناء كذا".

ستنفذ هذه الحلقة كتلة ال اكواد مرة واحدة، قبل التحقق مما إذا كان while. هي نسخة مختلفة من الحلقة do while الحلقة الشرط صحيحًا، ثم تكرر الحلقة طالما كان الشرط صحيحًا.

بناء الجملة

```
do {
 // code block to be executed
}
while (condition);
```

(✓ مثال ⇣⇣)

حلقة. سيتم دائمًا تنفيذ الحلقة مرة واحدة على الأقل، حتى لو كان الشرط خاطئًا، do while يستخدم ال(✓ مثال ⇣⇣) أدناه لأنه يتم تنفيذ كتلة ال اكواد قبل اختبار الشرط:

(✓ مثال ⇣⇣)

```
do {
  abib += "The number is " + i;
  i++;
}
while (i < 10);
```


لا تنس زيادة المتغير المستخدم في الشرط، وإلا فلن تنتهي الحلقة أبدًا!

مقارنة لحلقة بينما

مع حذف for، تشبه إلى حد كبير حلقة while فسوف تكتشف أن حلقة for. إذا كنت قد قرأت الفصل السابق، حول حلقة العبارة 1 والعبارة 3.

(✓ مثال ⇣⇣) تستخدم الحلقة في هذا ال for حلقة لجمع أسماء من مصفوفة :

(✓ مثال ⇣⇣)

```
const HaBiB= ["Abu Habib Al-Husini", "Hamza", "Habib", "Habib"];
let i = 0;
let abib_ = "";

for (;HaBiB[i];) {
  abib_ += HaBiB[i];
 i++;
}
```


(⇣⇣ مثال ✓) حلقة لجمع أسماء من مصفوفة while تستخدم الحلقة في هذا ال:

(✓ مثال ⇣⇣)

```
const HaBiB= ["Abu Habib Al-Husini", "Hamza", "Habib", "Habib"];
let i = 0;
let abib = "";

while (HaBiB[i]) {
  abib += HaBiB[i];
  i++;
}
```

العبارة "break" تخرج من الحلقة.

العبارة "continue" تقفز فوق تكرار واحد في الحلقة.

كلمة التوقف من داخل الحلقة

الكلمة switch() العبارة المستخدمة في فصل سابق من هذا الكتاب. تم استخدامه "للقفز" من break لقد رأيت بالفعل.

للانتقال من الحلقة break يمكن أيضًا استخدام العبارة:

(✓ مثال ⇣⇣)

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  abib += "The number is " + i + "<br>";
}
```

هو 3 (i) تنهي العبارة الحلقة ("تكسر" الحلقة) عندما يكون عداد الحلقة break في ال(✔ مثال ⇣⇣) أعلاه.

كلمة الاستمرار

تكسر العبارة continue تكرارًا واحدًا (في الحلقة)، في حالة حدوث شرط محدد، وتستمر مع التكرار التالي في الحلقة.

يتخطى هذا ال(✔ مثال ⇣⇣) قيمة 3:

(✔ مثال ⇣⇣)

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  abib += "The number is " + i + "<br>";
}
```

كتاب الكافى فى جافاسكربت الجزء الثانى ابو حبيب الحسينى

تسميات

لتسمية عبارات جافاسكربت، عليك أن تسبق العبارات باسم تسمية ونقطتين:

```
label:
statements
```

إن عبارات break و continue هي عبارات جافاسكربت الوحيدة التي يمكنها "القفز من فوق عنصر او خارج" الحلقة ال اكواد .

بناء الجملة:

```
break labelName;
```

```
continue labelName;
```

لا يمكن استخدام العبارة continue (مع أو بدون مرجع تسمية) إلا لتخطي تكرار حلقة واحدة .

لا يمكن استخدام العبارة break، التي لا تحتوي على مرجع تسمية، إلا للانتقال من حلقة أو مفتاح تبديل .

للانتقال من أي كتلة تعليمات برمجية Break باستخدام مرجع التسمية، يمكن استخدام عبارة:

(⇓⇓ مثال ✓)

```
const HaBiB= ["Abu Habib Al-Husini", "Hamza", "Habib", "Habib"];
list: {
  abib += HaBiB[0] + "<br>";
  abib += HaBiB[1] + "<br>";
  break list;
  abib += HaBiB[2] + "<br>";
  abib += HaBiB[3] + "<br>";
}
```


كتلة ال اكواد هي كتلة من ال اكواد بين { و }.

عناصر التكرارية

العناصر القابلة للتكرار هي كائنات قابلة للتكرار (مثل المصفوفات).

يمكن الوصول إلى العناصر القابلة للتكرار باستخدام تعليمات برمجية بسيطة وفعالة.

يمكن تكرار العناصر التكرارية باستخدام for..of الحلقات

للحلقة

تتكرر عبارة جافاسكربت for..of عبر عناصر الكائن القابل للتكرار.

بناء الجملة

```
for (variable of iterable) {
  // code block to be executed
}
```

التكرار

التكرار سهل الفهم.

إنه يعني ببساطة التكرار على نص من العناصر.

فيما يلي بعض الأمثلة السهلة:

- التكرار على نص
- التكرار على مصفوفة

التكرار على نص

يمكنك استخدام for..of حلقة للتكرار على عناصر النص:

(مثال ✓ ⇣⇣)

```
const Name = "Abu Habib alhosiny ";

for (const x of Name) {
  // code block to be executed
}
```


التكرار على مصفوفة

يمكنك استخدام for..of حلقة للتكرار على عناصر المصفوفة:

(مثال ✓ ⇣⇣)

```
const Husinie = ["a","b","c"];

for (const x of Husinie) {
  // code block to be executed
}
```

التكرار على مجموعة

يمكنك استخدام for..of حلقة للتكرار على عناصر المجموعة:

(مثال ✔ ⇣⇣)

```
const Husinie = new Set(["a","b","c"]);

for (const x of Husinie) {
  // code block to be executed
}
```


التكرار على خريطة المصفوفه

يمكنك استخدام for..of حلقة للتكرار على عناصر خريطة المصفوف :

(مثال ✔ ⇣⇣)

```
const HaBiB= new Map([
  ["Abo Habip", 500],
  ["Al-Husini__", 300],
  ["Osman", 200]
]);

for (const x of HaBiB) {
  // code block to be executed
}
```

مجموعات

مجموعة جافاسكربت عبارة عن مجموعة من القيم الفريدة.

يمكن أن تحدث كل قيمة مرة واحدة فقط في المجموعة.

طرق المجموعة المحجوزة

Method	وصف
new Set()	إنشاء مجموعة جديدة
add()	يضيف عنصرًا جديدًا إلى المجموعة
delete()	إزالة عنصر من مجموعة
has()	يُرجع صحيحًا في حالة وجود قيمة في المجموعة
forEach()	يستدعي رد اتصال لكل عنصر في المجموعة
values()	إرجاع مكرر بجميع القيم الموجودة في المجموعة
size	إرجاع عدد العناصر في المجموعة

كيفية إنشاء مجموعة

يمكنك إنشاء مجموعة جافاسكربت عن طريق:

- تمرير مصفوفة إلى new Set()
- قم بإنشاء مجموعة جديدة واستخدمها add() لإضافة القيم
- قم بإنشاء مجموعة جديدة واستخدمها add() لإضافة المتغيرات

وظيفة Set ()

تمرير مصفوفة إلى new Set() المنشئ:

(✔ مثال ⇓⇓)

```
// Create a Set
const Husinie = new Set(["a","b","c"]);
```


إنشاء مجموعة وإضافة القيم:

(✔ مثال ⇓⇓)

```
// Create a Set
const Husinie = new Set();

// Add Values to the Set
Husinie.add("a");
Husinie.add("b");
Husinie.add("c");
```


إنشاء مجموعة وإضافة المتغيرات:

(✔ مثال ⇓⇓)

```
// Create a Set
const Husinie = new Set();

// Create Variables
const a = "a";
const b = "b";
const c = "c";

// Add Variables to the Set
Husinie.add(a);
Husinie.add(b);
Husinie.add(c);
```

وظيفة الإضافة

(مثال ✔ ⇣⇣)

```
Husinie.add("d");
Husinie.add("e");
```


إذا أضفت عناصر متساوية، فسيتم حفظ العنصر الأول فقط:

(مثال ✔ ⇣⇣)

```
Husinie.add("a");
Husinie.add("b");
Husinie.add("c");
Husinie.add("c");
Husinie.add("c");
Husinie.add("c");
Husinie.add("c");
Husinie.add("c");
```

وظيفة forEach()

forEach() تستدعي الوظيفة دالة لكل عنصر مجموعة:

(مثال ✔ ⇣⇣)

```
// Create a Set
const Husinie = new Set(["a","b","c"]);

// List all Elements
let abib = "";
Husinie.forEach (function(value) {
  abib+= value;
})
```

وظيفة values()

تقوم الوظيفة بإرجاع كائن مكرر جديد يحتوي على جميع القيم الموجودة في المجموعة:

(مثال ✔ ⇣⇣)

```
Husinie.values()  // Returns [object Set Iterator]
```


يمكنك الآن استخدام كائن Iterator للوصول إلى العناصر:

(مثال ✔ ⇣⇣)

```
// List all Elements
let abib = "";
for (const x of Husinie.values()) {
abib += x;
}
```

خرائط

تحتوي خريطة المصفوف على أزواج قيمة المفتاح حيث يمكن أن تكون المفاتيح أي نوع كلمات.

تتذكر خريطة المصفوف ترتيب الإدراج الأصلي للمفاتيح.

كيفية إنشاء خريطة

يمكنك إنشاء خريطة جافاسكربت عن طريق:

- تمرير مصفوفة إلى new Map()
- إنشاء خريطة واستخدامها Map.set()

وظيفة new Map()

يمكنك إنشاء خريطة عن طريق تمرير مصفوفة إلى المُنشئ :

(مثال ✔ ⇓⇓)

```
// Create a Map
const HaBiB= new Map([
  ["Abo Habip", 500],
  ["Al-Husini__", 300],
  ["Osman", 200]
]);
```

وظيفة set()

يمكنك إضافة عناصر إلى خريطة المصفوف باستخدام الوظيفة:

(مثال ✔ ⇓⇓)

```
// Create a Map
const HaBiB= new Map();

// Set Map Values
HaBiB.set("Abo Habip", 500);
HaBiB.set("Al-Husini__", 300);
HaBiB.set("Osman", 200);
```


لتغيير قيم خريطة المصفوف الموجودة set() يمكن أيضًا استخدام الوظيفة:

(مثال ✔ ⇓⇓)

```
HaBiB.set("Abo Habip", 200);
```

وظيفة الحصول على قيمة

تحصل الوظيفة get() على قيمة المفتاح في خريطة المصفوف :

(مثال ✓ ⇣⇣)

```
HaBiB.get("Abo Habip");   // Returns 500
```

خاصية الحجم

تقوم الخاصية size بإرجاع عدد العناصر في خريطة المصفوف :

(مثال ✓ ⇣⇣)

```
HaBiB.size;
```

وظيفة الحذف

تقوم الوظيفة delete() بإزالة عنصر خريطة المصفوف :

(مثال ✓ ⇣⇣)

```
HaBiB.delete("Abo Habip");
```

الوظيفة has()

تُرجع الوظيفة has() صحيحًا في حالة وجود مفتاح في خريطة المصفوف :

(✓ مثال ⇣⇣)

```
HaBiB.has("Abo Habip");
```


جرب هذا:

```
HaBiB.delete("Abo Habip");
HaBiB.has("Abo Habip");
```

كائنات مقارنة بـ الخرائط

- الاختلافات بين كائنات جافاسكربت والخرائط:

خريطة	هدف
قابلة للتكرار مباشرة	غير قابلة للتكرار بشكل مباشر
لديك خاصية الحجم	ليس لديك خاصية الحجم
يمكن أن تكون المفاتيح أي نوع كلمات	يجب أن تكون المفاتيح عبارة عن نصوص (أو اكواد)
يتم ترتيب المفاتيح عن طريق الإدراج	المفاتيح ليست مرتبة بشكل جيد
ليس لديك مفاتيح افتراضية	لديك مفاتيح افتراضية

Iterable	Size	Key Types	Key Order	Defaults

وظيفة forEach()

forEach() تستدعي الوظيفة لكل زوج من المفاتيح/القيمة في خريطة المصفوف:

(✓ مثال ⇣⇣)

```
// List all entries
let abib = "";
HaBiB.forEach (function(value, key) {
abib += key + ' = ' + value;
})
```

وظيفة الإدخالات

تقوم الوظيفة entries() بإرجاع كائن مكرر مع [مفتاح العنصر] في خريطة المصفوف:

(⇣⇣ مثال ✓)

```
// List all entries
let abib = "";
for (const x of HaBiB.entries()) {
abib += x;
}
```


يوجد في جافاسكربت 5 أنواع مختلفة من الكلمات التي يمكن أن تحتوي على قيم:

- string
- number
- boolean
- object
- function

هناك 6 أنواع من الكائنات:

- Object
- Date
- Array
- String
- Number
- Boolean

ونوعان من الكلمات لا يمكن أن يحتويا على قيم:

- null
- undefined

نوع المشغل

يمكنك استخدام typeof عامل التشغيل للعثور على نوع كلمات متغير جافاسكربت.

(مثال ⇩⇩ ✓)

```
typeof "Habib"            // Returns "string"
typeof 3.14               // Returns "number"
typeof NaN                // Returns "number"
typeof false              // Returns "boolean"
typeof [1,2,3,4]          // Returns "object"
typeof {Name:'Habib', age:34}  // Returns "object"
typeof new Date()         // Returns "object"
typeof function () {}     // Returns "function"
typeof my$Habib           // Returns "undefined" *
typeof null               // Returns "object"
```


يرجى ملاحظة ما يلي:

- نوع كلمات NaN هو رقم
- نوع كلمات المصفوفة هو كائن
- نوع كلمات التاريخ هو كائن
- نوع الكلمات الفارغة هو كائن
- * نوع كلمات المتغير غير المحدد غير محدد
- * نوع كلمات المتغير الذي لم يتم تعيين قيمة له غير محدد أيضًا

لتحديد ما إذا كان كائن جافاسكربت عبارة عن مصفوفة (أو تاريخ) typeof لا يمكنك استخدامه.

الكلمات البدائية

قيمة الكلمات البدائية هي قيمة كلمات بسيطة واحدة بدون خصائص وطرق إضافية.

يمكن للمشغل typeof إرجاع أحد هذه الأنواع البدائية:

- string
- number
- boolean
- undefined

(مثال ✔ ⇓⇓)

```
typeof "Habib"          // Returns "string"
typeof 3.14             // Returns "number"
typeof true             // Returns "boolean"
typeof false            // Returns "boolean"
typeof x                // Returns "undefined" (if x has no value)
```

ايجاد النوع

يمكن للمشغل typeof إرجاع أحد النوعين :

- function
- object

يقوم عامل typeof التشغيل بإرجاع "كائن" للكائنات والمصفوفات والقيمة الخالية.

لا يقوم عامل typeof التشغيل بإرجاع "كائن" للوظائف.

(مثال ✔ ⇓⇓)

```
typeof {Name:'Habib', age:34} // Returns "object"
typeof [1,2,3,4]              // Returns "object" (not "array", see note below)
typeof null                   // Returns "object"
typeof function myFunc(){}    // Returns "function"
```


يقوم عامل typeof التشغيل بإرجاع " object" للمصفوفات لأن المصفوفات في جافاسكربت عبارة عن كائنات.

تابع typeof

المشغل typeof ليس لديهم أي نوع كلمات (+ - * /) ليس متغيرا. إنه عامل.

لكن typeof(تحتوي على نوع المعامل) يقوم دائمًا بإرجاع نص عامل التشغيل.

خاصية البناء

تقوم الخاصية constructor بإرجاع وظيفة المنشئ لجميع متغيرات جافاسكربت.

(✓ مثال ⇓⇓)

```
"Habib".constructor              // Returns function String()  {[native code]}
(3.14).constructor               // Returns function Number()  {[native code]}
false.constructor                // Returns function Boolean() {[native code]}
[1,2,3,4].constructor            // Returns function Array()   {[native code]}
{Name:'Habib',age:34}.constructor  // Returns function Object()  {[native code]}
new Date().constructor           // Returns function Date()    {[native code]}
function () {}.constructor       // Returns function Function(){[native code]}
```


يمكنك التحقق من خاصية المنشئ لمعرفة ما إذا كان الكائن Array (مصفوفة "):يحتوي على كلمة

(✓ مثال ⇓⇓)

```
function isArray(Husinie_Array) {
  return Husinie_Array.constructor.toString().indexOf("Array") > -1;
}
```


أو حتى بشكل أبسط، يمكنك التحقق مما إذا كان الكائن عبارة عن دالة Array :

(✓ مثال ⇓⇓)

```
function isArray(Husinie_Array) {
  return Husinie_Array.constructor === Array;
}
```

يحتوي على كلمة "تاريخ"(Date يمكنك التحقق من خاصية المنشئ لمعرفة ما إذا كان الكائن"):

(مثال ✓ ⇣⇣)

```
function isDate(myDate) {
  return myDate.constructor.toString().indexOf("Date") > -1;
}
```


أو حتى بشكل أبسط، يمكنك التحقق مما إذا كان الكائن عبارة عن دالة تاريخ :

(مثال ✓ ⇣⇣)

```
function isDate(myDate) {
  return myDate.constructor === Date;
}
```

غير معرف

النوع هو أيضا . undefine في جافاسكربت، المتغير بدون قيمة له القيمة .

(مثال ✓ ⇣⇣)

```
let $Habib;   // Value is undefined, type is undefined
```


سيكون النوع أيضًا undefine. يمكن إفراغ أي متغير عن طريق ضبط القيمة على undefine.

(مثال ✓ ⇣⇣)

```
$Habib= undefined;   // Value is undefined, type is undefined
```

القيم الفارغة

القيمة الفارغة ليس لها علاقة بـ undefine.

تحتوي النص الفارغة على قيمة قانونية ونوع.

(✓ مثال ⇣⇣)

```
let $Habib= "";   // The value is "", the typeof is "string"
```

خطا

يوجد "لا شيء". من المفترض أن يكون شيئًا غير موجود null في جافاسكربت.

هو كائن null لسوء الحظ، في جافاسكربت، نوع الكلمات.

كائن null. ينبغي أن يكون typeof null يمكنك اعتباره خطأً في جافاسكربت وهو

يمكنك إفراغ كائن عن طريق تعيينه على null:

(✓ مثال ⇣⇣)

```
let hAbiB = {firstName:"Habib", lastName:"Al Husini 😀😀😀", age:50, eyeColor:"blue"};
hAbiB = null;   // Now value is null, but type is still an object
```


يمكنك أيضًا إفراغ كائن عن طريق تعيينه على undefined:

(✓ مثال ⇣⇣)

```
let hAbiB = {firstName:"Habib", lastName:"Al Husini 😀😀😀", age:50, eyeColor:"blue"};
hAbiB = undefined;   // Now both value and type is undefined
```

الفرق بين غير محدد وفارغ

undefined ومتساويتان **null** في النوع ومختلفتان في القيمة:

```
typeof undefined        // undefined
typeof null             // object

null === undefined      // false
null == undefined       // true
```


مثيل المشغل

instanceof إذا كان الكائن مثيلًا للكائن المحدد **true** يُرجع عامل التشغيل:

(مثال ✓ ⇣⇣)

```
const HaBiB= ["Habib", "Hamza", "Abu Habib Al-Husini"];

(HaBiB instanceof Array);
(HaBiB instanceof Object);
(HaBiB instanceof String);
(HaBiB instanceof Number);
```


المشغل الفارغ

يقوم عامل التشغيل **void** بتقييم البرامتر وإرجاع غير محدد . غالبًا ما يُستخدم هذا العامل للحصول على القيمة الأولية غير المحددة، باستخدام **"void(0)"** (مفيد عند تقييم تعبير دون استخدام القيمة المرجعة).

(مثال ✓ ⇣⇣)

```
<a href="javascript:void(0);">
  Useless link
</a>
```

```
<a href="javascript:void(document.body.style.backgroundColor='red'>
  Click me to change the background color of body to red
</a>
```

تحويل نوع

يمكن تحويل متغيرات جافاسكربت إلى متغير جديد ونوع كلمات آخر:

- عن طريق استخدام وظيفة جافاسكربت
- تلقائيا عن طريق جافاسكربت نفسها

تحويل النصوص إلى أرقام

تقوم الوظيفة العامة Number() بتحويل متغير (أو قيمة) إلى رقم.

يتم تحويل نص رقمية (مثل "3.14") إلى رقم (مثل 3.14).

يتم تحويل نص فارغة (مثل "") إلى 0.

يتم تحويل نص غير رقمية (مثل "Habib") إلى NaN(ليس رقمًا).

أمثلة

سيتم تحويل هذه:

```
Number("3.14")
Number(Math.PI)
Number(" ")
Number("")
```

لن يتم تحويل هذه:

```
Number("99 88")
Number("Habib")
```

طرق ووظائف الأرقام

ستجد في الفصل **طرق الأرقام المزيد من الطرق التي يمكن استخدامها لتحويل النصوص إلى أرقام:**

Method	وصف
Number()	إرجاع رقم، تم تحويله من الوسيط الخاص به
parseFloat()	ياخذ نص ويعيد رقم الفاصلة العشرية
parseInt()	ياخذ نص ويعيد عددا صحيحا

المشغل الأحادي+

يمكن استخدام العامل الأحادي + لتحويل المتغير إلى رقم:للتذكير

(مثال ✓ ⇣⇣)

```
let y = "5";     // y is a string
let x = + y;     // x is a number
```


إذا تعذر تحويل المتغير، فسيظل رقمًا، ولكن بقيمة NaN (ليس رقمًا):

(مثال ✓ ⇣⇣)

```
let y = "Habib";  // y is a string
let x = + y;      // x is a number (NaN)
```

كيف تحويل الأرقام إلى نصوص

String() يمكن للوظيفة العامة تحويل الأرقام إلى نصوص.

يمكن استخدامه على أي نوع من الأرقام أو القيم الحرفية أو المتغيرات أو البرامترات:

(✓ مثال ⇣⇣)

```
String(x)       // returns a string from a number variable x
String(123)     // returns a string from a number literal 123
String(100 + 23)  // returns a string from a number from an expression
```


وظيفة الرقم toString() تفعل الشيء نفسه.

(✓ مثال ⇣⇣)

```
x.toString()
(123).toString()
(100 + 23).toString()
```

المزيد من الدوال

ستجد في الفصل طرق الأرقام المزيد من الطرق التي يمكن استخدامها لتحويل الأرقام إلى نصوص فى كتاب المرجع الكامل للجافاسكربت:

Method	وصف
toExponential()	إرجاع نص برقم مقرب ومكتوب باستخدام التدوين الأسي.
toFixed()	تُرجع نص تحتوي على رقم مقرب ومكتوب بعدد محدد من الكسور العشرية.
toPrecision()	إرجاع نص برقم مكتوب بطول محدد

تحويل التواريخ إلى أرقام

لتحويل التواريخ إلى أرقام Number() يمكن استخدام الوظيفة العامة.

```
d = new Date();
Number(d)       // returns 1404568027739
```

وظيفة التاريخ getTime() تفعل الشيء نفسه.

```
d = new Date();
abibDate.getTime()      // returns 1404568027739
```

تحويل التواريخ إلى نصوص

تحويل التواريخ إلى نصوص String() يمكن للوظيفة العامة.

```
String(Date()) // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"
```

وظيفة التاريخ toString() تفعل الشيء نفسه.

(✔ مثال ⇓⇓)

```
Date().toString() // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"
```

ستجد في فصل طرق التاريخ المزيد من الطرق التي يمكن استخدامها لتحويل التواريخ إلى نصوص:

Method	وصف
getDate()	احصل على اليوم كرقم (1-31)
getDay()	احصل على رقم أيام الأسبوع (0-6)
getFullYear()	احصل على السنة المكونة من أربعة أرقام (yyyy)
getHours()	احصل على الساعة (0-23)
getMilliseconds()	احصل على الملي ثانية (0-999)
getMinutes()	احصل على الدقائق (0-59)
getMonth()	احصل على الشهر (0-11)
getSeconds()	احصل على الثواني (0-59)
getTime()	احصل على الوقت (ملي ثانية منذ 1 يناير 1970)

تحويل القيم المنطقية إلى أرقام

أيضًا تحويل القيم المنطقية إلى أرقام Number() يمكن للوظيفة العامة.

```
Number(false)    // returns 0
Number(true)     // returns 1
```

تحويل القيم المنطقية إلى نصوص

تحويل القيم المنطقية إلى نصوص String() يمكن للوظيفة العامة.

```
String(false)    // returns "false"
String(true)     // returns "true"
```

الوظيفة المنطقية toString() تفعل الشيء نفسه.

```
false.toString()  // returns "false"
true.toString()   // returns "true"
```

التحويل التلقائي للنوع

عندما تحاول جافاسكربت العمل على نوع كلمات "خاطئ"، ستحاول تحويل القيمة إلى نوع "صحيح".

والنتيجة ليست دائما ما تتوقعه:

```
5 + null   // returns 5         because null is converted to 0
"5" + null  // returns "5null"   because null is converted to "null"
"5" + 2    // returns "52"      because 2 is converted to "2"
"5" - 2    // returns 3         because "5" is converted to 5
"5" * "2"   // returns 10        because "5" and "2" are converted to 5 and 2
```

تحويل النص التلقائي

عندما تحاول "إخراج" كائن أو متغير toString() تقوم جافاسكربت تلقائيًا باستدعاء وظيفة المتغير:

```
document.getElementById("Habib").innerHTML = myVar;
```

```
// if abibVar= {Name:"FHabib"}  // toString converts to "[object Object]"
// if abibVar= [1,2,3,4]       // toString converts to "1,2,3,4"
```

```
// if abibVar= new Date()     // toString converts to "Fri Jul 18 2014 09:08:55
GMT+0200"
```

يتم أيضًا تحويل الأرقام والقيم المنطقية، لكن هذا غير مرئي تمامًا:

```
// if abibVar= 123          // toString converts to "123"
// if abibVar= true         // toString converts to "true"
// if abibVar= false        // toString converts to "false"
```

جدول تحويل الانواع فى

يوضح هذا الجدول نتيجة تحويل قيم جافاسكربت المختلفة إلى Boolean وString وNumber:

القيمة	التحويل الى رقم	التحويل الى نص	التوحويل الى بولين
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	true
"000"	0	"000"	true
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true
""	0	""	false
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[ ]	0	""	true
[20]	20	"20"	true
[10,20]	NaN	"10,20"	true
["twenty"]	NaN	"twenty"	true
["ten","twenty"]	NaN	"ten,twenty"	true
function(){}	NaN	"function(){}"	true

{}	NaN	"[object Object]"	true
null	0	"null"	false
undefined	NaN	"undefined"	false

. تشير القيم الموجودة بين علامتي الاقتباس إلى قيم النص

تشير القيم الحمراء إلى قيم (بعض) قد لا يتوقعها المبرمجون.

Bitwise عمليات جافاسكربت

Bitwise مشغلي

المعامل	الاسم	وصف
&	AND	يضبط كل بت على 1 إذا كان كلا البتين 1
|	OR	يضبط كل بت على 1 إذا كان أحد البتين هو 1
^	XOR	يضبط كل بت على 1 إذا كان واحد فقط من البتتين هو 1
~	NOT	يعكس كل البتات
<<	Zero fill left shift	ينتقل إلى اليسار عن طريق دفع الأصفار من اليمين وترك البتات الموجودة في أقصى اليسار تسقط
>>	Signed right shift	ينتقل إلى اليمين عن طريق دفع نسخ من أقصى اليسار إلى الداخل من اليسار، ويترك البتات الموجودة في أقصى اليمين تسقط
>>>	Zero fill right shift	ينتقل إلى اليمين عن طريق دفع الأصفار من اليسار، ويترك البتات الموجودة في أقصى اليمين تسقط

أمثلة

المعامل	النتيجة	مثال	النتيجة
5 & 1	1	0101 & 0001	0001
5 | 1	5	0101 | 0001	0101
~ 5	10	~0101	1010
5 << 1	10	0101 << 1	1010
5 ^ 1	4	0101 ^ 0001	0100
5 >> 1	2	0101 >> 1	0010
5 >>> 1	2	0101 >>> 1	0010

يستخدم معاملات 32 بت

ملحوظة

اذا كنت لا تعرف لغة البايناري التى يفهمها الكمبيوتر لا تقرا هذا الفصل لانك لن تفهم شى اقراء كتاب تعلم لغة البيناري بالعربى

اولا تقوم جافاسكربت بتخزين الأرقام كأرقام فاصلة عائمة بطول 64 بت، ولكن يتم تنفيذ جميع العمليات المتعلقة بالبت على أرقام ثنائية بطول 32 بت.

قبل تنفيذ عملية البت، تقوم جافاسكربت بتحويل الأرقام إلى أعداد صحيحة ذات 32 بت.

بعد إجراء عملية البت، يتم تحويل النتيجة مرة أخرى إلى أرقام جافاسكربت بحجم 64 بت.

تستخدم الأمثلة أعلاه أرقامًا ثنائية غير موقعة مكونة من 4 بتات. وبسبب هذا ~ 5 يعود 10.

بما أن جافاسكربت تستخدم أعدادًا صحيحة مكونة من 32 بت، فلن تُرجع 10. وستُرجع -6

00000000000000000000000101 (5)

11111111111111111111111111111010 (~5 = -6)

يستخدم العدد الصحيح الموقّع الجزء الموجود في أقصى اليسار كعلامة الطرح.

Bitwise AND

عند تنفيذ عملية AND 1 على زوج من البتات، فإنها تُرجع 1 إذا كانت كلا البتتين.

التحويل للبيت

مثال ✓ (⇓⇓ بت 4:

عملية	نتيجة
1111 | 0000	1111
1111 | 0001	1111
1111 | 0010	1111
1111 | 0100	1111

كتاب الكافى فى جافاسكربت ال

Bitwise XOR

عندما يتم تنفيذ XOR على زوج من البتات، فإنه يُرجع 1 إذا كانت البتات مختلفة:

مثال (✓ ⇓⇓) 4 بت:

عملية	نتيجة
1111 ^ 0000	1111
1111 ^ 0001	1110
1111 ^ 0010	1101
1111 ^ 0100	1011

Bitwise AND (&)

Bitwise AND يُرجع 1 فقط إذا كانت كلا البتتين 1:

عدد عشري	الثنائية
5	00000000000000000000000101
1	00000000000000000000000001
5 و 1	00000000000000000000000001 (1)

مثال (✓ ⇓⇓)

```
let x = 5 & 1;
```


Bitwise أو (|)

Bitwise OR 1 يُرجع 1 إذا كانت إحدى البتات هي:

عدد عشري	الثنائية
5	00000000000000000000000101
1	00000000000000000000000001
5 | 1	00000000000000000000000101 (5)

مثال (✓ ⇓⇓)

```
let x = 5 | 1;
```

Bitwise XOR (^)

يقوم **Bitwise XOR** بإرجاع **1** إذا كانت البتات مختلفة:

عدد عشري	الثنائية
5	00000000000000000000000101
1	00000000000000000000000001
5 ^ 1	00000000000000000000000100 (4)

(مثال ✓ ⇣⇣)

```
let x = 5 ^ 1;
```


Bitwise NOT (~)

عدد	الثنائية
5	00000000000000000000000101
~5	11111111111111111111111111111010 (-6)

(مثال ✓ ⇣⇣)

```
let x = ~5;
```


كيف إزاحة البت لليسار بمقدار (<<)

هذا هو التحول الأيسر ملء الصفر. يتم دفع بت واحد أو أكثر من الصفر من اليمين، وتسقط البتات الموجودة في أقصى اليسار:

عدد	الثنائية
5	00000000000000000000000101
5 << 1	00000000000000000000001010 (10)

(مثال ✓ ⇣⇣)

```
let x = 5 << 1;
```

علامة الإزاحة الى اليمين(>>)

هذه علامة تحافظ على التحول الصحيح. يتم دفع نسخ من البت الموجود في أقصى اليسار إلى الداخل من اليسار، وتسقط البتات الموجودة في أقصى اليمين:

عدد عشري	الثنائية
-5	11111111111111111111111111111011
-5 >> 1	11111111111111111111111111111101 (-3)

(مثال ✓ ⇣⇣)

```
let x = -5 >> 1;
```


علامة التحول الأيمن (>>>)

هذا هو التحول الأيمن ملء الصفر. يتم دفع بت واحد أو أكثر من الصفر من اليسار، وتسقط البتات الموجودة في أقصى اليمين:

عدد عشري	الثنائية
5	00000000000000000000000101
5 >>> 1	00000000000000000000000010 (2)

(مثال ✓ ⇣⇣)

```
let x = 5 >>> 1;
```


ترجمة الارقام الى لغة البينارى الثنائية

من السهل فهم الأرقام الثنائية التي تحتوي على مجموعة بت واحدة فقط:

الترجمة الثنائي	القيمة
00000000000000000000000001	1
00000000000000000000000010	2
00000000000000000000000100	4

00000000000000000000001000	8
00000000000000000000010000	16
00000000000000000000100000	32
00000000000000000001000000	64

يؤدي تعيين عدد قليل من البتات إلى الكشف عن الخاصية الثنائي:

الترجمة الثنائي	القيمة
00000000000000000000000101	5 (4 + 1)
00000000000000000000001101	13 (8 + 4 + 1)
00000000000000000000101101	45 (32 + 8 + 4 + 1)

يتم تخزين أرقام جافاسكربت الثنائية بتنسيق مكمل ثنائي.

هذا يعني أن الرقم السالب هو رقم البت وليس الرقم زائد 1:

الترجمة الثنائي	القيمة
00000000000000000000000101	5
11111111111111111111111111111011	-5
00000000000000000000000110	6
11111111111111111111111111111010	-6
00000000000000000000101000	40
11111111111111111111111111011000	-40

تحويل عشري إلى ثنائي

(✓ مثال ⇣⇣)

```
function habib2bin(dec){
  return (dec >>> 0).toString(2);
}
```

تحويل ثنائي إلى عشري

(✓ مثال ⇣⇣)

```
function habib_bin2dec(bin){
  return parseInt(bin, 2).toString(10);
}
```

البرامترات العادية

البرامتر العادي عبارة عن نص من الأحرف التي تشكل نمط بحث.

يمكن استخدام نمط البحث في عمليات البحث عن النص واستبدال النص.

ما هو البرامتر العادي؟

البرامتر العادي عبارة عن نص من الأحرف التي تشكل نمط البحث .

عندما تبحث عن كلمات في نص ما، يمكنك استخدام نمط البحث هذا لوصف ما تبحث عنه.

يمكن أن يكون البرامتر العادي حرفًا واحدًا أو نمطًا أكثر تعقيدًا.

يمكن استخدام البرامترات العادية لإجراء جميع أنواع عمليات البحث عن النص واستبدال النص .

قوانين البحث

(⇣⇣ مثال ✓)

```
/Abu Habib alhosiny /i;
```

شاهد ال(⇣⇣ مثال ✓):

/Abu Habib alhosiny /i هو تعبير عادي.

() هو نمط (يستخدم في البحث).

i هو مُعدِّل (يعدل البحث ليكون غير حساس لحالة الأحرف).

استخدام دوال النص

في جافاسكربت، غالبًا ما تُستخدم البرامترات العادية مع طريقتي النص : search()وreplace().

تستخدم الوظيفة search()تعبيرًا للبحث عن تطابق، وتقوم بإرجاع موضع التطابق.

تقوم الوظيفة replace()بإرجاع نص معدلة حيث يتم استبدال الخاصية.

طرق البحث فى النصوص

تبحث الوظيفة search() عن نص لقيمة محددة وترجع موضع المطابقة:

(⇣⇣ مثال ✓)

استخدم نص لإجراء بحث عن "Abu Habib alhosiny " في نص :

```
let abib = "Visit Abu Habib alhosiny !";
let n = abib.search("Abu Habib alhosiny ");
```

ستكون n النتيجة في:

6

احد طرق البحث الغير حساس لحالة الاحرف

(مثال ✔ ⇓⇓)

استخدم تعبيرًا عاديًا لإجراء بحث غير حساس لحالة الأحرف عن "Abu Habib alhosiny " في نص :

```
let abib = "Visit Abu Habib alhosiny ";
let n = abib.search(/Abu Habib alhosiny /i);
```

ستكون النتيجة في n:

6

كيف البحث و الاستبدال

تستبدل الوظيفة replace() قيمة محددة بقيمة أخرى في نص :

```
let abib = "Visit Hamza!";
let result = abib.replace("Hamza", "Abu Habib alhosiny ");
```

طرق اخرى الاستبدال الغير حساس للحالة

(مثال ✔ ⇓⇓)

استخدم تعبيرًا عاديًا غير حساس لحالة الأحرف لاستبدال Hamza بـ Abu Habib alhosiny في نص :

```
let abib = "Visit Hamza!";
let result = abib.replace(/Hamza/i, "Abu Habib alhosiny ");
```

النتيجة في المكون ستكون:

```
//Visit Abu Habib alhosiny !
```

هل لاحظت؟

يمكن استخدام وسيطات البرامتر العادي (بدلاً من وسيطات النص) في الطرق المذكورة أعلاه.
يمكن أن تجعل البرامترات العادية بحثك أكثر قوة (غير حساس لحالة الأحرف على سبيل ال(✓ مثال ⇣⇣))

معدّلات البرامتر العادي

يمكن استخدام المعدلات لإجراء المزيد من عمليات البحث العامة غير الحساسة لحالة الأحرف:

Modifier	وصف
i	إجراء مطابقة غير حساسة لحالة الأحرف
g	إجراء مطابقة عامة (البحث عن كافة التطابقات بدلاً من التوقف بعد الاجراء الأولى)
m	إجراء مطابقة متعددة الأسطر

خصائص البرامتر العادية

تُستخدم الأقواس للعثور على مجموعة من الأحرف:

Expression	وصف
[abc]	ابحث عن أي من الأحرف الموجودة بين قوسين
[0-9]	ابحث عن أي من الأرقام الموجودة بين القوسين
(x|y)	ابحث عن أي من البدائل مفصولة بـ |

الأحرف الأولية هي أحرف ذات معنى خاص:

Metacharacter	وصف
\d	ابحث عن رقم
\s	ابحث عن حرفمسافة فارغة
\b	ابحث عن تطابق في بداية كلمة مثل هذه: \bWORD، أو في نهاية كلمة مثل هذه: WORD\b
\uxxxx	ابحث عن حرف Unicode المحدد بالرقم السداسي العشري xxxx

تحدد الكميات الكميات:

Quantifier	وصف
n+	يطابق أي نص تحتوي على واحد على الأقل n
n*	يطابق أي نص تحتوي على صفر أو أكثر من تكرارات n
n?	يطابق أي نص تحتوي على صفر أو تكرار واحد لـ n

RegExp باستخدام كائن

هو كائن تعبير عادي له خصائص ودوال محددة مسبقًا RegExp في جافاسكربت، كائن سنتحدث عنة ان شاء الله.

اساليب الاختبار

الوظيفة test() هي وظيفة تعبير RegExp.

يبحث في نص عن نمط، ويعيد صواب أو خطأ، اعتمادًا على النتيجة.

يبحث ال "e": التالي عن نص للحرف (✓ مثال ⇣⇣)

(✓ مثال ⇣⇣)

```
const pattern = /e/;
pattern.test("Abu Habeb Al Husene *_*    ___");
```

بما أن هناك حرف "e" في النص ، فإن مخرجات الكود أعلاه ستكون:

true

ليس عليك وضع البرامتر العادي في متغير أولاً. يمكن اختصار السطرين أعلاه إلى سطر واحد:

```
/e/.test("Abu Habib Al Husini *_*  ___");
```

تابع طرق البحث

الوظيفة exec() هي وظيفة تعبير RegExp.

يبحث في نص عن نمط محدد، ويعيد النص الذي تم العثور عليه ككائن.

إذا لم يتم العثور على أي تطابق، فسيتم إرجاع كائن فارغ (فارغ) .

يبحث ال "e": التالي عن نص للحرف (✓ مثال ⇣⇣)

(✓ مثال ⇣⇣)

```
/e/.exec("Abu Habeb Al Husene *_*  ___");
```

أسبقية المشغل

تصف أسبقية عامل التشغيل الترتيب الذي يتم به تنفيذ العمليات في البرامتر الحسابي.

الضرب (*) والقسمة (/) لهما أسبقية أعلى من الجمع (+) والطرح (-).

كما هو الحال في الرياضيات التقليدية، يتم الضرب أولاً:

```
let x = 100 + 50 * 3;
```


عند استخدام الأقواس، يتم حساب العمليات داخل الأقواس أولاً:

```
let x = (100 + 50) * 3;
```


يتم حساب العمليات التي لها نفس الأولوية (مثل * و/) من اليسار إلى اليمين:

```
let x = 100 / 50 * 3;
```


التعامل مع الأخطاء فى جافاسكربت

التقاط الخطا دون مشاكل

جملة تراى التى سنستخدمها الان للامساك بالخطاء وكشفها وارساله فهى تحاول تنفيذ الكود فان نفذته خير وبركة وان لم تنفذة تمسك الخطا وترسله

كتلة ال اكواد للتشغيل (للمحاولة) try يحدد الكلمة

كتلة ال اكواد للتعامل مع أي خطأ catch يحدد الكلمة.

كتلة ال اكواد للتشغيل بغض النظر عن النتيجة finally يحدد الكلمة.

خطأ مخصص throw يحدد الكلمة.

عند حدوث الخطا

عند تنفيذ كود جافاسكربت، يمكن أن تحدث أخطاء مختلفة.

يمكن أن تكون الأخطاء عبارة عن أخطاء ترميزية يقوم بها المبرمج، وأخطاء ناتجة عن إدخال خاطئ، وأشياء أخرى غير متوقعة.

(⇣⇣ مثال ✓)

لإنتاج خطأ عمدًا لا يوجد شى اسمة **"adddlert"** الاسم الصحيح **"alert"** في هذا ال(✓ مثال ⇣⇣)، أخطأنا في كتابة :

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

<script>
try {
  adddlert("Welcome Habib*_*!");
}
catch(err) {
  document.getElementById("Habib").innerHTML = err.message;
}
</script>
```


تكتشف جافاسكربت خطأ **Adddler** معه وتنفذ رمز الالتقاط للتعامل معه ، وتنفذ ت.

عبيلة واديلة 😂

لا تستغرب من هذ العنوان لانه يصف جملة تراى اذا كتبت كل ال اكواد بداخلها ستجمع لك جميع الاخطا وترسلها اليك . لتعرفها

تحديد كتلة من ال اكواد ليتم اختبارها بحثًا عن الأخطاء أثناء تنفيذها **try** يتيح لك الكلمة

تحديد كتلة من ال اكواد التي سيتم تنفيذها، في حالة حدوث خطأ في كتلة المحاولة **catch** يتيح لك الكلمة.

عبارات جافاسكربت **try** وتأتي **catch** في أزواج:

```
try {
  Block of code to try
}
catch(err) {
```

```
  Block of code to handle errors
}
```

ماذا يحدث عند الاخطاء

عند حدوث خطأ، ستتوقف جافاسكربت عادةً وتنتج رسالة خطأ.

المصطلح الفني لذلك هو: سوف يقوم جافاسكربت بطرح استثناء (رمي خطأ).

ستقوم جافاسكربت فعليًا بإنشاء كائن Error ونوع الخطا والرسالة والاسم : بخاصيتين .

التحكم فى الاخطاء

يسمح لك الكلمة throw بإنشاء خطأ مخصص.

من الناحية الفنية يمكنك طرح استثناء (رمي خطأ).

يمكن أن يكون الاستثناء جافاسكربت: Object أو a أو Boolean أو Numbera أو String:

```
throw "Too big";   // throw a abib
throw 500;         // throw a number
```

إذا كنت تستخدم throw مع try و catch، فيمكنك التحكم في تدفق البرنامج وإنشاء رسائل خطأ مخصصة.

للتحقق من صحة الإدخال

يفحص هذا ال(✓ مثال ⇣⇣) المدخلات. إذا كانت القيمة خاطئة، فسيتم طرح استثناء (يخطئ).

بواسطة عبارة الالتقاط ويتم عرض رسالة خطأ مخصصة (err) يتم اكتشاف الاستثناء:

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>

<input id="Habib" type="abib">
<button type="button" onclick="Husini()">Test Input</button>
<p id="habib3"></p>
```

```
<script>
function Husini() {
  const message = document.getElementById("habib3");
  message.innerHTML = "";
  let x = document.getElementById("Habib").value;
  try {
    if(x.trim() == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```

التحقق من صحة HTML

الكود أعلاه هو مجرد (✍ مثال ⇣⇣).

المضمن، وذلك باستخدام قواعد HTML غالبًا ما تستخدم المتصفحات الحديثة مزيجًا من جافاسكربت والتحقق من صحة التحقق المحددة مسبقًا والمحددة في سمات HTML:

```
<input id="Habib" type="number" min="5" max="10" step="1">
```

يمكنك قراءة المزيد حول التحقق من صحة النماذج في فصل لاحق من هذا الكتاب.

البرامتر الاخير

تنفيذ ال اكواد ، بعد المحاولة والالتقاط، بغض النظر عن النتيجةfinally تتيح لك العبارة:

بناء الجملة

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

(⇓⇓ مثال ✓)

```
function Husini() {
  const message = document.getElementById("habib3");
  message.innerHTML = "";
  let x = document.getElementById("Habib").value;
  try {
    if(x.trim() == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Error: " + err + ".";
  }
  finally {
    document.getElementById("Habib").value = "";
  }
}
```

كائن الخطأ

يحتوي جافاسكربت على كائن خطأ مضمن يوفر معلومات الخطأ عند حدوث خطأ.

يوفر كائن الخطأ خاصيتين مفيدتين: الاسم والرسالة.

قيم اسم الخطأ

يمكن إرجاع ست قيم مختلفة بواسطة خاصية اسم الخطأ:

يتم وصف القيم الست المختلفة أدناه.

خطأ في التقييم

يشير EvalError إلى خطأ في وظيفة eval().

بدلاً من ذلك SyntaxError استخدم EvalError. الإصدارات الأحدث من جافاسكربت لا ترمي.

خطأ في النطاق

إذا كنت تستخدم رقمًا يقع خارج نطاق القيم القانونية RangeError يتم طرح A.

على سبيل ال(✓ مثال ⇣⇣): لا يمكنك تعيين عدد الأرقام المهمة لرقم ما على 500.

(✓ مثال ⇣⇣)

```
let num = 1;
try {
  num.toPrecision(500);  // A number cannot have 500 significant digits
}
catch(err) {
  document.getElementById("Habib").innerHTML = err.Name;
}
```

خطأ مرجعي

إذا كنت تستخدم (مرجعًا) متغيرًا لم يتم الإعلان عنه ReferenceError يتم طرح A:

(✔ مثال ⇓⇓)

```
let x = 5;
try {
  x = y + 1;  // y cannot be used (referenced)
}
catch(err) {
  document.getElementById("Habib").innerHTML = err.Name;
}
```

خطأ في بناء الجملة

إذا حاولت تقييم ال اكواد التي تحتوي على خطأ في بناء الجملة.SyntaxError يتم طرح A

(✔ مثال ⇓⇓)

```
try {
  eval("alert('Easy-to)");  // Missing ' will produce an error
}
catch(err) {
  document.getElementById("Habib").innerHTML = err.Name;
}
```

كيف معرفة نوع الخطا

إذا كنت تستخدم قيمة خارج نطاق الأنواع المتوقعة:TypeError يتم طرح A

(✔ مثال ⇓⇓)

```
let num = 1;
try {
  num.toUpperCase();  // You cannot convert a number to upper case
}
catch(err) {
```

```
  document.getElementById("Habib").innerHTML = err.Name;
}
```

كيف معرفة اخطا الروابط

يتم طرح A URIError إذا كنت تستخدم أحرفًا غير قانونية في دالة URI:

(مثال ✔ ⇓⇓)

```
try {
  decodeURI("%%%");  // You cannot URI decode percent signs
}
catch(err) {
  document.getElementById("Habib").innerHTML = err.Name;
}
```

نطاق الكتلة

قبل ES6 (2015)، كان لجافاسكربت نطاق عام ونطاق وظيفي فقط.

قدم ES6 كلمتين رئيسيتين جديدتين مهمتين في جافاسكربت: let و const.

توفر هاتان الكلمتان الرئيسيتان نطاق الكتلة في جافاسكربت.

لا يمكن الوصول إلى المتغيرات المعلنة داخل الكتلة { } من خارج الكتلة:

(مثال ✔ ⇓⇓)

```
{
  let x = 2;
```

```
}
// x can NOT be used here
```

الكلمة المحجوزة أن تحتوي على نطاق حظر var لا يمكن للمتغيرات المعلنة باستخدام.

يمكن الوصول إلى المتغيرات المعلنة داخل الكتلة { } من خارج الكتلة.

(مثال ✓ ⇣⇣)

```
{
  var x = 2;
}
// x CAN be used here
```

النطاق المحلي

المتغيرات المعلنة داخل وظيفة جافاسكربت، تصبح محلية للوظيفة.

(مثال ✓ ⇣⇣)

```
// code here can NOT use hAbiB4

function Husini() {
  let hAbiB4 = "Hamza";
  // code here CAN use hAbiB4
}

// code here can NOT use hAbiB4
```


المتغيرات المحلية لها نطاق الوظيفة :

ولا يمكن الوصول إليها إلا من داخل الوظيفة.

وبما أنه يتم التعرف على المتغيرات المحلية فقط داخل وظائفها، فيمكن استخدام المتغيرات التي تحمل الاسم نفسه في وظائف مختلفة.

يتم إنشاء المتغيرات المحلية عند بدء تشغيل الدالة، ويتم حذفها عند اكتمال الدالة.

نطاق الوظيفة

لدى جافاسكربت نطاق وظيفي: كل وظيفة تنشئ نطاقًا جديدًا.

لا يمكن الوصول إلى المتغيرات المحددة داخل الوظيفة (مرئية) من خارج الوظيفة.

المتغيرات المعلنة بـ var، let تكون const متشابهة تمامًا عند الإعلان عنها داخل دالة.

لديهم جميعا نطاق الوظيفة :

```
function Husini() {
  var hAbiB4 = "Hamza";  // Function Scope
}
```

```
function Husini() {
  let hAbiB4 = "Hamza";  // Function Scope
}
```

```
function Husini() {
  const hAbiB4 = "Hamza";  // Function Scope
}
```

متغيرات العامة

المتغير المعلن خارج الدالة يصبح GLOBAL .

(مثال ⇓⇓ ✓)

```
let hAbiB4 = "Hamza";
// code here can use hAbiB4

function Husini() {
// code here can also use hAbiB4
}
```


المتغير العام له نطاق عام :

يمكن لجميع الكود والوظائف الموجودة على صفحة الويب الوصول إليها.

النطاق العام

المتغيرات المعلنة عالميًا (خارج أي وظيفة) لها نطاق عام .

يمكن الوصول إلى المتغيرات العامة من أي مكان في برنامج جافاسكربت.

المتغيرات المعلنة بـ var، let تكون constمتشابهة تمامًا عندما يتم الإعلان عنها خارج الكتلة.

لديهم جميعا نطاق عام :

```
var x = 2;       // Global scope

let x = 2;       // Global scope

const x = 2;     // Global scope
```

متغيرات

في جافاسكربت، الكائنات والوظائف هي أيضًا متغيرات.

يحدد النطاق إمكانية الوصول إلى المتغيرات والكائنات والوظائف من أجزاء مختلفة من ال اكواد .

العامة التلقائيه

إذا قمت بتعيين قيمة لمتغير لم يتم التصريح عنه، فسوف يصبح تلقائيًا متغيرًا عالميًا .

حتى لو تم تعيين القيمة داخل دالة، hAbiB4 سيعلن (✓ مثال ⇣⇣) ال اكواد هذا عن متغير عام

(✓ مثال ⇣⇣)

```
Husini();

// code here can use hAbiB4

function Husini() {
```

```
 hAbiB2= "Hamza";
}
```

الوضع الصارم

".تدعم جميع المتصفحات الحديثة تشغيل جافاسكربت في "الوضع الصارم

.في "الوضع الصارم"، لا تصبح المتغيرات غير المعلنة عمومية تلقائيًا

المتغيرات العامة في HTML

.مع جافاسكربت، النطاق العام هو بيئة جافاسكربت

.في HTML، النطاق العام هو كائن النافذة

:المتغيرات العامة المحددة بالكلمة var المحجوزة تنتمي إلى كائن النافذة

(⇣⇣ مثال ✓)

```
var hAbiB4 = "Hamza";
// code here can use window.hAbiB4
```


:المتغيرات العامة المحددة بالكلمة let المحجوزة لا تنتمي إلى كائن النافذة

(⇣⇣ مثال ✓)

```
let hAbiB4 = "Hamza";
// code here can not use window.hAbiB4
```

تحذير

لا تقم بإنشاء متغيرات عمومية إلا إذا كنت تقصد ذلك.

يمكن للمتغيرات العامة (أو الوظائف) أن تحل محل متغيرات النافذة (أو الوظائف)
يمكن لأي وظيفة، بما في ذلك كائن النافذة، استبدال المتغيرات والوظائف العامة.

عمر المتغيرات

يبدأ عمر متغير جافاسكربت عند الإعلان عنه.

يتم حذف متغيرات الوظيفة (المحلية) عند اكتمال الوظيفة.

في متصفح الويب، يتم حذف المتغيرات العامة عند إغلاق نافذة (أو علامة التبويب) المتصفح.

البرمترات فى الوظائف

تعمل وسيطات الدالة (البرمترات) كمتغيرات محلية داخل الوظائف.

اعلان عن متغير

في جافاسكربت، يمكن الإعلان عن المتغير بعد استخدامه.

بعبارة أخرى؛ يمكن استخدام المتغير قبل الإعلان عنه.

يعطي ال(✔ مثال ⇣⇣) 1 نفس نتيجة ال(✔ مثال ⇣⇣) 2 :

1 (⇣⇣ مثال ✔)

```
x = 5; // Assign 5 to x

elem = document.getElementById("Habib"); // Find an element
elem.innerHTML = x;                      // Display x in the element

var x; // Declare x
```

(✓ مثال ⇣⇣) 2

```
var x; // Declare x
x = 5; // Assign 5 to x

elem = document.getElementById("Habib"); // Find an element
elem.innerHTML = x;                      // Display x in the element
```


لفهم هذا، عليك أن تفهم مصطلح "الرفع".

الرفع هو السلوك الافتراضي لجافاسكربت المتمثل في نقل كافة الإعلانات إلى أعلى النطاق الحالي (إلى أعلى الكود الحالي أو الوظيفة الحالية).

افضليات لـ const و Let

المتغيرات التي تم تعريفها let ورفعها const إلى أعلى الكتلة، ولكن لم تتم تهيئتها.

المعنى: كتلة ال اكواد على علم بالمتغير، لكن لا يمكن استخدامها حتى يتم الإعلان عنها.

سيؤدي استخدام متغير قبل الإعلان عنه إلى let ReferenceError.

المتغير موجود في "منطقة متروكة مؤقتة" من بداية الكتلة حتى يتم الإعلان عنه:

(✓ مثال ⇣⇣)

سيؤدي هذا إلى ReferenceError:

```
hAbiB4 = "Hamza";
let hAbiB4;
```


يعد استخدام const متغير قبل الإعلان عنه خطأً في بناء الجملة، لذا لن يتم تشغيل ال اكواد ببساطة.

(✓ مثال ⇣⇣)

لن يتم تشغيل هذا الرمز.

```
hAbiB4 = "Hamza";
const hAbiB4;
```

لا يتم رفع عمليات التهيئة

تقوم جافاسكربت برفع الإعلانات فقط، وليس عمليات التهيئة.

ال(✔ مثال ⇣⇣) 1 لا يعطي نفس نتيجة ال(✔ مثال ⇣⇣) 2 :

1 (⇣⇣ مثال ✔)

```
var x = 5; // Initialize x
var y = 7; // Initialize y

elem = document.getElementById("Habib"); // Find an element
elem.innerHTML = x + " " + y;          // Display x and y
```


2 (⇣⇣ مثال ✔)

```
var x = 5; // Initialize x

elem = document.getElementById("Habib"); // Find an element
elem.innerHTML = x + " " + y;          // Display x and y

var y = 7; // Initialize y
```


هل يعقل أن y غير محدد في ال(✔ مثال ⇣⇣) الأخير؟

وذلك لأنه يتم رفع الإعلان (var y) إلى الأعلى (7=) فقط، وليس التهيئة.

غير محددة y قبل استخدامه، ولكن نظرًا لعدم رفع عمليات التهيئة، فإن قيمة y بسبب الرفع، تم الإعلان عن.

ال(✔ مثال ⇣⇣) 2 هو نفس الكتابة:

(✔ مثال ⇣⇣)

```
var x = 5; // Initialize x
var y;     // Declare y

elem = document.getElementById("Habib"); // Find an element
elem.innerHTML = x + " " + y;          // Display x and y

y = 7;   // Assign 7 to y
```

أعلن عن المتغيرات في الأعلى!

يعد الرفع (بالنسبة للعديد من المطورين) سلوكًا غير معروف أو يتم التغاضي عنه في جافاسكربت.

إذا لم يفهم المطور الرفع، فقد تحتوي البرامج على أخطاء (أخطاء).

لتجنب الأخطاء، قم دائمًا بإعلان كافة المتغيرات في بداية كل نطاق.

نظرًا لأن هذه هي الوظيفة التي تفسر بها جافاسكربت ال اكواد ، فهي دائمًا قاعدة جيدة.

لا تسمح جافاسكربت في الوضع الصارم باستخدام المتغيرات إذا لم يتم الإعلان عنها.

استخدم الكلمة Strict

يحدد أنه يجب تنفيذ تعليمات جافاسكربت البرمجية في "الوضع الصارم";"use strict".

توجيه "الاستخدام الصارم".

كان التوجيه جديدًا في الإصدار 5 من ECMAScript "use strict".

إنه ليس عبارة، ولكنه تعبير حرفي، تم تجاهله في الإصدارات السابقة من جافاسكربت.

هو الإشارة إلى أنه يجب تنفيذ ال اكواد في "الوضع الصارم" "use strict". والغرض من ذلك

مع الوضع الصارم، لا يمكنك، على سبيل ال(✔ مثال ⇣⇣)، استخدام متغيرات غير معلنة.

تدعم كافة المتصفحات الحديثة "الاستخدام الصارم" باستثناء 9 والإصدارات الأقدم:

تحدد الأرقام الموجودة في الجدول إصدار المتصفح الأول الذي يدعم التوجيه بشكل كامل.

يمكنك استخدام الوضع الصارم في جميع برامجك. يساعدك على كتابة تعليمات برمجية أنظف، مثل منعك من استخدام متغيرات غير معلنة.

"use strict" بإلقاء خطأ حتى لو لم يفهمه IE 9 هي مجرد نص ، لذلك لن يقوم.

إعلان الوضع الصارم

يتم الإعلان عن الوضع الصارم بإضافة "استخدام صارم"؛ هذة معنى الكلمة إلى بداية الكود أو الوظيفة.

تم الإعلان عنه في بداية الكود، وله نطاق عام (سيتم تنفيذ جميع ال اكواد الموجودة في الكود في الوضع الصارم):

(مثال ⇓⇓ ✓)

```
"use strict";
x = 3.14;      // This will cause an error because x is not declared
```


(مثال ⇓⇓ ✓)

```
"use strict";
Husini();

function Husini() {
  y = 3.14;  // This will also cause an error because y is not declared
}
```


تم الإعلان عنها داخل دالة، ولها نطاق محلي (فقط الكود الموجود داخل الوظيفة هو في الوضع الصارم):

```
x = 3.14;      // This will not cause an error.
Husini();

function Husini() {
  "use strict";
```

```
  y = 3.14;  // This will cause an error
}
```

تابع استخدام الوضع الصارم

تم تصميم بناء الجملة، للإعلان عن الوضع الصارم، ليكون متوافقًا مع الإصدارات الأقدم من جافاسكربت.

لانها كلمة مكتوبة مثل هذة الجملة في ليس له أي آثار . ("Abu Bakr Al-Siddiq"؛) تجميع حرفي أو نص حرفية . ببساطة يجمع إلى متغير غير موجود و لا يحدث خطا فى المتصفحات القديمة التى لا تعرف شى عن الوضع الصارم

لذلك "use strict";معناه لا يهم إلا المتصفحات الجديده الذين.

لماذا الوضع الصارم؟

الوضع الصارم يجعل من السهل كتابة جافاسكربت "الآمنة".

يغير الوضع الصارم "بناء الجملة السيئ" المقبول مسبقًا إلى أخطاء حقيقية.

على سبيل ال(✓ مثال ⇣⇣)، في جافاسكربت العادية، يؤدي الخطأ في كتابة اسم المتغير إلى إنشاء متغير عام جديد. في الوضع الصارم، سيؤدي هذا إلى حدوث خطأ، مما يجعل من المستحيل إنشاء متغير عام عن طريق الخطأ.

في جافاسكربت العادية، لن يتلقى المطور أي تعليقات خطأ في تعيين قيم لخصائص غير قابلة للكتابة.

فقط، أو خاصية غير موجودة، أو متغير غير getter في الوضع الصارم، أي إسناد إلى خاصية غير قابلة للكتابة، أو خاصية موجود، أو كائن غير موجود، سيؤدي إلى حدوث خطأ.

الغير مسموح به في الوضع الصارم

لا يجوز استخدام المتغير دون الإعلان عنه:

```
"use strict";
x = 3.14;          // This will cause an error
```


الكائنات هي متغيرات أيضا.

لا يجوز استخدام كائن دون الإعلان عنه:

```
"use strict";
x = {p1:10, p2:20};     // This will cause an error
```


غير مسموح بحذف متغير (أو كائن).

```
"use strict";
let x = 3.14;
delete x;           // This will cause an error
```


غير مسموح بحذف وظيفة.

```
"use strict";
function x(p1, p2) {};
delete x;           // This will cause an error
```


لا يُسمح بتكرار اسم المعلمة:

```
"use strict";
function x(p1, p1) {};   // This will cause an error
```


غير مسموح بالأحرف الرقمية الثماني:

```
"use strict";
let x = 010;          // This will cause an error
```


غير مسموح بأحرف الاستثناء الثمانية:

```
"use strict";
let x = "\010";         // This will cause an error
```


الكتابة إلى خاصية للقراءة فقط غير مسموح بها:

```
"use strict";
const obj = {};
Object.defineProperty(obj, "x", {value:0, writable:false});

obj.x = 3.14;          // This will cause an error
```


لا يُسمح بالكتابة إلى خاصية الحصول عليها فقط:

```
"use strict";
const obj = {get x() {return 0} };

obj.x = 3.14;          // This will cause an error
```


لا يُسمح بحذف خاصية غير قابلة للحذف:

```
"use strict";
delete Object.prototype; // This will cause an error
```


لا يمكن استخدام الكلمة eval كمتغير:

```
"use strict";
let eval = 3.14;       // This will cause an error
```

لا يمكن استخدام الكلمة arguments كمتغير:

```
"use strict";
let arguments = 3.14;   // This will cause an error
```


لا يجوز الكلمة with :

```
"use strict";
with (Math){x = cos(2)}; // This will cause an error
```


لا يسمح بإنشاء متغيرات في النطاق الذي تم استدعاؤه منه eval()، لأسباب أمنية.

في الوضع الصارم، لا يمكن استخدام المتغير قبل الإعلان عنه:

```
"use strict";
eval ("x = 2");
alert (x);     // This will cause an error
```


الإعلان عن متغير باستخدام الكلمة المحجوزة eval() في الوضع الصارم، لا يمكن لـ var:

```
"use strict";
eval ("var x = 2");
alert (x);    // This will cause an error
```


الإعلان عن متغير باستخدام الكلمة المحجوزة eval() لا يمكن لـ Let:

```
eval ("let x = 2");
alert (x);      // This will cause an error
```


المحجوزة في الوظائف بشكل مختلف في الوضع الصارم this تتصرف الكلمة.

المحجوزة إلى الكائن الذي يسمى الوظيفة this تشير الكلمة.

وستعيد الوظائف في الوضع العادي الكائن العام undefined إذا لم يتم تحديد الكائن، فستعود الوظائف في الوضع الصارم (النافذة):

```
"use strict";
function Husini() {
  alert(this); // will alert "undefined"
}
Husini();
```

تابع الوضع الصارم

لا يمكن استخدام الكلمات المحجوزة لإصدارات جافاسكربت المستقبلية كأسماء متغيرات في الوضع الصارم.

```
"use strict";
let public = 1500;     // This will cause an error
```


!احترس

يتم التعرف على التوجيه "استخدام صارم" فقط في بداية البرنامج النصي أو الوظيفة.

هذه الكلمة المحجوزة

(مثال ✔ ⬇⬇)

```
const hAbiB = {
  firstName: "Habib",
  lastName : "Al Husini 😀😀😀",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

الكلمة زيس

. تشير الكلمة المحجوزة إلى كائن this، في جافاسكربت

. استدعائه (استخدامه أو استدعائه) this يعتمد الكائن على كيفية

المحجوزة إلى كائنات مختلفة اعتمادًا على كيفية استخدامها this تشير الكلمة:

ملحوظة

this. ليس متغيرا. إنها كلمة محجوزة . لا يمكنك تغيير قيمة this

---

الإشارة من داخل الدالة

. فإنه يشير إلى الكائن this، عند استخدامه في وظيفة كائن

. يشير إلى كائن العنصر this، في ال(مثال ✔ ⬇⬇) الموجود أعلى هذه الصفحة

. لأن وظيفة الاسم الكامل هي وظيفة لكائن العنصر

```
fullName : function() {
  return this.firstName + " " + this.lastName;
}
```

تابع كلمة زيس

this يشير إلى الكائن العام، عند استخدامه بمفرده .

this يعمل في النطاق العام لأنه .

[object Window]: الكائن العام في نافذة المتصفح هو

(مثال ⇓⇓ ✓)

```
let x = this;
```


this يشير أيضًا إلى الكائن العام، في الوضع الصارم ، عند استخدامه بمفرده :

(مثال ⇓⇓ ✓)

```
"use strict";
let x = this;
```

كلمة هذا فى الوضع العادى

this في إحدى الوظائف، يكون الكائن العام هو الرابط الافتراضي لـ.

[object Window]: الكائن العام في نافذة المتصفح هو

(مثال ⇓⇓ ✓)

```
function Husini() {
  return this;
}
```

كلمة هذا فى الوضع الصارم

لا يسمح الوضع الصارم لجافاسكربت بالربط الافتراضي.

لذا، عند استخدامه في دالة، في الوضع الصارم، this يكون undefine.

(مثال ✓ ⇣⇣)

```
"use strict";
function Husini() {
  return this;
}
```

كتاب الكافى فى جافاسكربت الجزء الثانى ابو حبيب الحسينى

كلمة هذا في معالجات الأحداث

في معالجات أحداث HTML، this يشير إلى عنصر HTML الذي استقبل الحدث:

(مثال ✓ ⇣⇣)

```
<button onclick="this.style.display='none'">
  Click to Remove Me!
</button>
```

كتاب الكافى فى جافاسكربت الجزء الثانى ابو حبيب الحسينى

ربط وظيفة الكائن

في هذه الأمثلة، this هو العنصر الكائن :

(✔ مثال ⇣⇣)

```
const hAbiB = {
  firstName  : "Habib",
  lastName   : "Al Husini 😀😁😂",
  id         : 5566,
  Husini : function() {
    return this;
  }
};
```


(✔ مثال ⇣⇣)

```
const hAbiB = {
  firstName: "Habib",
  lastName : "Al Husini 😀😁😂",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```


هو خاصية الاسم الأول لهذا (كائن العنصر) this.firstName، على سبيل ال(✔ مثال ⇣⇣).

ربط الوظيفة الصريحة

هي دوال جافاسكربت محددة مسبقًا apply() والدوال call() الدوال.

يمكن استخدامهما لاستدعاء أسلوب كائن مع كائن آخر كوسيطة.

أنظر أيضا:

حتى لو كان hAbiB2، كوسيطة، ويشير هذا إلى hAbiB2 مع hAbiB1.fullName يستدعي ال(✔ مثال ⇣⇣ أدناه) fullName تابعًا لـ hAbiB1:

(⇣⇣ مثال ✓)

```
const hAbiB1 = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}

const hAbiB2 = {
  firstName:"Habib",
  lastName: "Al Husini 😀😁😂",
}


hAbiB1.fullName.call(hAbiB2);

// Return "Habib Al Husini 😀😁😂":
```

تابع وظائف الكائن

الوظيفة، يمكن للكائن استعارة وظيفة من كائن آخر bind() باستخدام.

يقوم هذا ال (✓ مثال ⇣⇣) بإنشاء كائنين (عنصر وعضو).

يستعير كائن العضو وظيفة الاسم الكامل من كائن العنصر:

(⇣⇣ مثال ✓)

```
const hAbiB = {
  firstName:"Habib",
  lastName: "Al Husini 😀😁😂",
  fullName: function () {
    return this.firstName + " " + this.lastName;
  }
}

const member = {
  firstName:"Habib",
```

```
  lastName: "Nilsen",
}

let fullName = hAbiB.fullName.bind(member);
```

الوظيفة المختصرة

تم تقديم وظائف مختصرة في ES6.

تسمح لنا الدوال مختصرة بكتابة صيغة دالة أقصر:

```
let Husini = (a, b) => a * b;
```


تابع الوظائف المختصرة :

```
Easy-to = function() {
  return "Abu Habib Al Husini *_*!";
}
```


وظيفة مختصرة اخرى:

```
Easy-to = () => {
  return "Abu Habib Al Husini *_*!";
}
```

يصبح أقصر! إذا كانت الدالة تحتوي على عبارة واحدة فقط، وكانت العبارة تُرجع قيمة، فيمكنك إزالة الأقواس والكلمة المحجوزة return:

الدوال المختصرة تُرجع القيمة افتراضيًا:

```
Easy-to = () => "Abu Habib Al Husini *_*!";
```


ملاحظة: يعمل هذا فقط إذا كانت الدالة تحتوي على عبارة واحدة فقط.

إذا كان لديك معلمات، يمكنك تمريرها داخل الأقواس:

الوظيفة المختصرة مع البرمترات:

```
Easy-to = (val) => "Easy-to " + val;
```


في الواقع، إذا كان لديك معلمة واحدة فقط، فيمكنك تخطي الأقواس أيضًا:

دالة المختصرة بدون الأقواس:

```
Easy-to = val => "Easy-to " + val;
```


---

this فى الوظائف المحتصرة

يختلف التعامل أيضًا في وظائف الأمختصرة مقارنة بالوظائف العاديةthis.

باختصار، مع وظائف الأمختصرة لا يوجد ربط لـ this.

تمثل الكلمة المحجوزة الكائن الذي يسمى الوظيفة، والذي يمكن أن يكون النافذة أو المستند أو الزر this. في الوظائف العادية أو أي شيء آخر.

this المحجوزة دائمًا الكائن الذي يحدد وظيفة المختصرة مع وظائف الأمختصرة ، تمثل الكلمة .

دعونا نلقي نظرة على المثالين لفهم الفرق.

كلا المثالين يستدعيان الوظيفة مرتين، أولاً عند تحميل الصفحة، ومرة أخرى عندما ينقر المستخدم على زر.

يستخدم ال(مثال) الأول دالة عادية، ويستخدم ال(مثال) الثاني دالة مختصرة.

توضح النتيجة أن ال✔ مثال الأول يُرجع كائنين مختلفين (النافذة والزر)، بينما يُرجع المثال الثاني كائن النافذة مرتين، لأن كائن النافذة هو "مالك" الدالة.

(✔ مثال ⇣⇣)

this يمثل الكائن الذي يستدعي الوظيفة مع وظيفة عادية:

```
// Regular Function:
Easy-to = function() {
  document.getElementById("Habib").innerHTML += this;
}

// The window object calls the function:
window.addEventListener("load", Easy-to);

// A button object calls the function:
document.getElementById("HAbib_btn").addEventListener("click", Easy-to);
```


(✔ مثال ⇣⇣)

this تمثل منشا الوظيفة مع دالة مختصرة:

```
// Arrow Function:
Easy-to = () => {
  document.getElementById("Habib").innerHTML += this;
}

// The window object calls the function:
window.addEventListener("load", Easy-to);
```

```
// A button object calls the function:
document.getElementById("HAbib_btn").addEventListener("click", Easy-to);
```


تذكر هذه الاختلافات عند العمل مع الوظائف. في بعض الأحيان يكون سلوك الوظائف العادية هو ما تريده، وإذا لم يكن الأمر كذلك، فاستخدم وظائف الأمختصرة .

بناء الكلاسات

لإنشاء كلاس class استخدم الكلمة المحجوزة .

أضف دائمًا وظيفة باسم constructor():

بناء الجملة

```
class ClassHabib {
  constructor() { ... }
}
```

(مثال ✔ ⇣⇣)

```
class $Habib{
  constructor(Name, year) {
    this.Name = Name;
    this.year = year;
  }
}
```

أعلاه بإنشاء كلاس تسمى (مثال ✔ ⇣⇣) يقوم ال "$Habib".

يحتوي الفصل او الكلاس على خاصيتين أوليتين: "الاسم" و"السنة".

كلاس جافاسكربت ليست كائنًا .

إنه قالب لكائنات جافاسكربت.

كيف استخدام الكلاس المنشا

عندما يكون لديك كلاس ، يمكنك استخدامه لإنشاء كائنات:

(⇣⇣ مثال ✓)

```
let my$Habib1 = new $Habib("Habib", 2014);
let my$Habib2 = new $Habib("Omar ibn Al-khattab", 2019);
```


يستخدم ال(✓ مثال ⇣⇣) أعلاه كلاس لإنشاء كائنين قيمة .

يتم استدعاء أسلوب المنشئ تلقائيًا عند إنشاء كائن جديد.

وظيفة البناء للكلاس او الكلاس

وظيفة البناء هي وظيفة خاصة:

- يجب أن يكون له الاسم الدقيق "المنشئ"
- يتم تنفيذه تلقائيًا عند إنشاء كائن جديد
- يتم استخدامه لتهيئة خصائص الكائن

إذا لم تحدد وظيفة منشئة، فستضيف جافاسكربت وظيفة منشئة فارغة.

طرق الكلاس

يتم إنشاء دوال الكلاس او الكلاس بنفس بناء الجملة مثل دوال الكائن.

استخدم الكلمة المحجوزة class لإنشاء كلاس .

قم دائمًا بإضافة وظيفة constructor().

ثم أضف أي عدد من الطرق.

بناء الجملة

```
class ClassHabib {
  constructor() { ... }
```

```
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
}
```

أنشئ وظيفة كلاس باسم "حبيب"،والتي تُرجع الاسم والسنة :

(✔ مثال ⇣⇣)

```
class $Habib{
  constructor(Name, year) {
    this.Name = Name;
    this.year = year;
  }
  age() {
    let date = new Date();
    return date.getFullYear() - this.year;
  }
}

let my$Habib= new $Habib("Habib", 2014);
document.getElementById("Habib").innerHTML =
"My $Habibis " + my$Habib.age() + " years olabibDate.";
```


يمكنك إرسال البرمترات إلى دوال الفصل او الكلاس:

(✔ مثال ⇣⇣)

```
class $Habib{
  constructor(Name, year) {
    this.Name = Name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}

let date = new Date();
let year = date.getFullYear();
```

```
let my$Habib= new $Habib("Habib", 2014);
document.getElementById("Habib").innerHTML=
"My $Habibis " + my$Habib.age(year) + " years olabibDate.";
```

وحدات جافاسكربت

الوحدات

تسمح لك وحدات جافاسكربت بتقسيم ال اكواد إلى ملفات منفصلة.

وهذا يجعل من السهل الحفاظ على قاعدة ال اكواد .

تعتمد وحدات جافاسكربت على العبارات **export** و**import**.

كيف تصدير الاجرائات

يمكنك تصدير دالة أو متغير من أي ملف.

دعونا ننشئ ملفًا باسم **hAbiB.js** ونملأه بالأشياء التي نريد تصديرها.

هناك نوعان من عمليات التصدير: المحددة والافتراضية.

تسمية التصدير

يمكنك إنشاء عمليات تصدير محددة بطريقتين. في الخط بشكل فردي، أو كلها مرة واحدة في الأسفل.

في الخط بشكل فردي:

hAbiB.js

```
export const Name = "hAbiB";
export const age = 40;
```

الكل مرة واحدة في الأسفل:

hAbiB.js

```
const Name = "hAbiB";
const age = 40;

export {Name, age};
```

التصديرات الافتراضية

ونستخدمه لتوضيح التصدير الافتراضي، habib.js دعونا ننشئ ملفًا آخر، اسمه

يمكنك الحصول على تصدير افتراضي واحد فقط في الملف.

(مثال ⇣⇣ ✔)

habib.js

```
const message = () => {
const Name = "hAbiB";
const age = 40;
return Name + ' is ' + age + 'years olabibDate.';
};

export default message;
```

كيف الاستيراد

يمكنك استيراد الوحدات الخاصية إلى ملف بطريقتين، بناءً على ما إذا كانت تسمى التصديرات أو الاصلرات الافتراضية.

يتم إنشاء التصديرات المحددة باستخدام الأقواس المتعرجة. التصديرات الافتراضية ليست كذلك.

كيف الاستيراد من التصديرات المنشاة

استيراد الاجرائات المحددة من الملف **hAbiB.js:**

```
import { Name, age } from "./hAbiB.js";
```


الاستيراد من التصديرات الافتراضية

استيراد تصدير افتراضي من الملف **habib.js:**

```
import message from "./habib.js";
```


ملحوظة

تعمل الوحدات فقط مع بروتوكول **HTTP(s).**

جافاسكربت جيسون

ما هو جسون؟

•

- JSON هو تنسيق قاعدة بيانات خفيف الوزن
- JSON * مستقلة عن اللغة
- JSON سهل الفهم

جيسون (مثال)

JSON (كائنات) تحدد صيغة كائن الموظفين: مصفوفة من 3 سجلات موظفين:

جيسون (⬇️⬇️ مثال ✔)

```
{
"employees":[
 {"firstName":"Habib", "lastName":"Al Husini 😀😁😂"},
 {"firstName":"Mohssen", "lastName":"Hassan"},
 {"firstName":"Habib", "lastName":"Fareed"}
]
}
```

لكائنات JSON يتم تقييم تنسيق

من الناحية النحوية مع ال اكواد لإنشاء كائنات جافاسكربت JSON يتطابق تنسيق.

بسهولة إلى كائنات جافاسكربت أصلية JSON وبسبب هذا التشابه، يمكن لبرنامج جافاسكربت تحويل كلمات.

قواعد بناء جملة JSON

- يتم فصل بين القيمة والمفتاح بفواصل
- الأقواس المتعرجة تجزء مجموعات القيم والمفاتيح
- الأقواس المربعة تحمل المصفوفات

اسم وقيمة - JSON كلمات

كأزواج اسم/قيمة، تمامًا مثل خصائص كائن جافاسكربت JSON تتم كتابة كلمات.

يتكون زوج الاسم/القيمة من اسم الحقل (بين علامتي اقتباس مزدوجتين)، متبوعًا بنقطتين، متبوعة بقيمة:

```
"firstName":"Habib"
```

كائنات JSON

تتم كتابة كائنات JSON داخل الأقواس المتعرجة.

تمامًا كما هو الحال في جافاسكربت، يمكن أن تحتوي الكائنات على أزواج متعددة من الأسماء/القيم:

```
{"firstName":"Habib", "lastName":"Al Husini 😀😁😂"}
```

مصفوفات جيسون

تتم كتابة مصفوفات JSON بين قوسين مربعين.

تمامًا كما هو الحال في جافاسكربت، يمكن أن تحتوي المصفوفة على كائنات:

```
"employees":[
 {"firstName":"Habib", "lastName":"Al Husini 😀😁😂"},
 {"firstName":"Mohssen", "lastName":"Hassan"},
 {"firstName":"Habib", "lastName":"Fareed"}
]
```

في ال(✔ مثال ⬇⬇) أعلاه، الكائن "المستخدمين" عبارة عن مصفوفة. أنه يحتوي على ثلاثة كائنات.

كل كائن هو سجل لعنصر (بالاسم الأول واسم العائلة).

تحويل JSON

هو قراءة الكلمات من خادم الويب وعرض الكلمات في صفحة الويب **JSON** الاستخدام الشائع لـ.

للتبسيط، يمكن إثبات ذلك باستخدام نص كمدخل.

أولاً، أنشئ نص جافاسكربت تحتوي على صيغة **JSON:**

```
let abib = '{ "employees" : [' +
'{ "firstName":"Habib" , "lastName":"Al Husini 😁😁😁" },' +
'{ "firstName":"Mohssen" , "lastName":"Hassan" },' +
'{ "firstName":"Habib" , "lastName":"Fareed" } ]}';
```

لتحويل النص إلى كائن جافاسكربت **JSON.parse()** بعد ذلك، استخدم وظيفة جافاسكربت المضمنة:

```
const obj = JSON.parse(abib);
```

وأخيرًا، استخدم كائن جافاسكربت الجديد في صفحتك:

(مثال ⇣⇣ ✓)

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

```
<script>
document.getElementById("Habib").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```


تصحيح أخطاء جافاسكربت

يمكن أن تحدث أخطاء في كل مرة تكتب فيها بعض اكواد الكمبيوتر الجديدة.

وظيفة console.log()

لعرض قيم جافاسكربت في نافذة مصحح console.log() إذا كان المتصفح يدعم تصحيح الأخطاء، فيمكنك استخدامه الأخطاء:

(مثال ✓ ⇣⇣)

```
<!DOCTYPE html>
<html>
<body>

<h1>Thank God The Lord Of Arabics</h1>

<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>

</body>
</html>
```


تحديد نقاط التوقف

في نافذة مصحح الأخطاء، يمكنك تعيين نقاط التوقف في كود جافاسكربت.

عند كل نقطة توقف، سيتوقف تنفيذ جافاسكربت، ويتيح لك فحص قيم جافاسكربت.

بعد فحص القيم، يمكنك استئناف تنفيذ الاكواد (عادةً باستخدام زر التشغيل).

الكلمة المحجوزة للمصحح

توقف الكلمة **debugger**(إذا كانت متوفرة) وتستدعي وظيفة تصحيح الأخطاء ، وتستدعي تنفيذ الكود المحجوزة.

هذا له نفس وظيفة تحديد نقطة توقف في مصحح الأخطاء.

في حالة عدم توفر أي تصحيح، لن يكون لعبارة مصحح الأخطاء أي تأثير.

عند تشغيل مصحح الأخطاء، سيتوقف تنفيذ هذا الرمز قبل تنفيذ السطر الثالث.

(مثال ⇓⇓ ✓)

```
let x = 15 * 5;
debugger;
document.getElementById("Habib").innerHTML = x;
```

هل كنت تعلم ؟

تصحيح الأخطاء هو عملية اختبار الأخطاء (الأخطاء) في برامج الكمبيوتر وإيجادها وتقليلها.
كان أول خطأ حاسوبي معروف هو خلل حقيقي (حشرة) عالق في الأجهزة الإلكترونية.

ملحوظة

ضع دائمًا مسافات حول عوامل التشغيل (= + - * /) وبعد الفواصل:

أمثلة:

```
let x = y + z;
const Husinie_Array = ["Hamza", "Habib", "Mohamed"];
```

المسافة البادئة للكود

استخدم دائمًا مسافتين لوضع مسافة بادئة لكتل ال اكواد :

المهام:

```
functionhapip(bib) {
  return (5 / 9) * (bib - 32);
}
```

للمسافة البادئة. يقوم المحررون المختلفون بتفسير علامات التبويب بشكل (tableators) لا تستخدم علامات التبويب مختلف.

قواعد

القواعد العامة للعبارات البسيطة:

- قم دائمًا بإنهاء العبارة البسيطة بفاصلة منقوطة.

أمثلة:

```
const HaBiB= ["Hamza", "Habib", "Mohamed"];
```

```
const hAbiB = {
  firstName: "Habib",
  lastName: "Al Husini 😀😁😂",
  age: 50,
  eyeColor: "blue"
};
```

القواعد العامة للكلمات المعقدة (المركبة):

- ضع قوس الفتح في نهاية السطر الأول.

•استخدم مسافة واحدة قبل قوس الفتح.
•ضع قوس الإغلاق على سطر جديد، بدون مسافات بادئة.
•

المهام:

```
functionhapip(bib) {
  return (5 / 9) * (bib - 32);
}
```

طرق تهيئة المتغيرات والبيانات

```
// Declare and initiate at the beginning
let firstName = "";
let lastName = "";
let hAbiB3= 0;
let discount = 0;
let fullhAbiB3= 0,
const Husinie_Array = [];
const myObject = {};
```

توفر تهيئة المتغيرات فكرة عن الاستخدام المقصود (ونوع الكلمات المقصودة).

الطرق البديلة

•استخدم "" بدلا منnew String()
•استخدم 0 بدلا منnew Number()
•استخدم false بدلا منnew Boolean()
•استخدم {} بدلا منnew Object()
•استخدم [] بدلا منnew Array()
•استخدم /()/ بدلا منnew RegExp()
•استخدم function (){} بدلا منnew Function()

(⇩⇩ مثال ✓)

```
let x1 = "";          // new primitive string
let x2 = 0;           // new primitive number
let x3 = false;       // new primitive boolean
const x4 = {};        // new object
const x5 = [];        // new array object
const x6 = /()/;      // new regexp object
const x7 = function(){}; // new function object
```

التكملة فى الجزء الثالث

باذن الله تعالى