Bu sayfada JavaScript'in neler yapabileceğine dair bazı örnekler yer almaktadır.

JavaScript HTML İçeriğini Değiştirebilir

Birçok JavaScript HTML yönteminden biri `getElementById()`.

Aşağıdaki örnek, bir HTML öğesini (id='demo' ile) "bulur" ve öğe içeriğini (innerHTML) "Merhaba JavaScript" olarak değiştirir:

Örnek

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

Kendin dene "

JavaScript hem çift hem de tek tırnak işaretlerini kabul eder:

Örnek

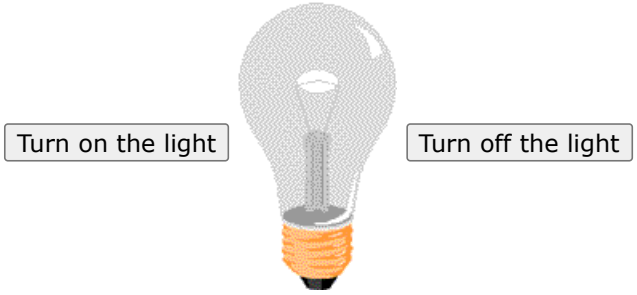
```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

Kendin dene "

JavaScript, HTML Özellik Değerlerini Değiştirebilir

`src` Bu örnekte JavaScript bir etiketin (source) özelliğinin değerini değiştirir `<img>`:

Ampul



Kendin dene "

JavaScript HTML Stillerini Değiştirebilir (CSS)

Bir HTML öğesinin stilini değiştirmek, bir HTML niteliğini değiştirmenin bir çeşididir:

Örnek

```
document.getElementById("demo").style.fontSize = "35px";
```

Kendin dene "

JavaScript HTML Öğelerini Gizleyebilir

HTML öğelerinin gizlenmesi stil değiştirilerek yapılabilir `display`:

Örnek

```
document.getElementById("demo").style.display = "none";
```

Kendin dene "

JavaScript HTML Öğelerini Gösterebilir

Gizli HTML öğelerinin gösterilmesi stil değiştirilerek de yapılabilir `display`:

Örnek

```
document.getElementById("demo").style.display = "block";
```

Kendin dene "

Biliyor musun?

JavaScript ve Java , hem konsept hem de tasarım açısından tamamen farklı dillerdir.

JavaScript, 1995 yılında Brendan Eich tarafından icat edildi ve 1997 yılında ECMA standardı haline geldi.

ECMA-262 standardın resmi adıdır. ECMAScript dilin resmi adıdır.

JavaScript Sürümleri »

JavaScript Nereye

‹ Öncesi Sonraki ›

<script> Etiketi

HTML'de `<script>` ve `</script>` etiketleri arasına JavaScript kodu eklenir.

Örnek

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

Kendin dene "

Eski JavaScript örnekleri bir tür niteliği kullanabilir: <script type="text/javascript">.
Type niteliği gerekli değildir. JavaScript, HTML'deki varsayılan kodlama dilidir.

JavaScript İşlevleri ve Olayları

JavaScript `function`, "çağrıldığında" çalıştırılabilen bir JavaScript kodu bloğudur.

Örneğin, kullanıcının bir düğmeyi tıklatması gibi bir **olay** meydana geldiğinde bir işlev çağrılabilir .

Daha sonraki bölümlerde işlevler ve olaylar hakkında daha fazlasını öğreneceksiniz.

<head> veya <body>'de JavaScript

Bir HTML belgesine istediğiniz sayıda komut dosyası yerleştirebilirsiniz.

Komut dosyaları bir HTML sayfasının `<body>`, veya bölümüne ya da her ikisine de yerleştirilebilir . `<head>`

<head>'de JavaScript

Bu örnekte, bir HTML sayfasının bölümüne bir JavaScript `function` yerleştirilmiştir . `<head>`

Bir düğmeye tıklandığında işlev çağrılır (çağrılır):

Örnek

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
```

```
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

Kendin dene "

<body>'de JavaScript

Bu örnekte, bir HTML sayfasının bölümüne bir JavaScript `function` yerleştirilmiştir . `<body>`

Bir düğmeye tıklandığında işlev çağrılır (çağrılır):

Örnek

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

<h2>Demo JavaScript in Body</h2>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Kendin dene "

Komut dosyalarının <body> öğesinin altına yerleştirilmesi görüntüleme hızını artırır çünkü komut dosyası yorumlaması ekranı yavaşlatır.

Harici JavaScript

Komut dosyaları ayrıca harici dosyalara da yerleştirilebilir:

Harici dosya: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

Aynı kod birçok farklı web sayfasında kullanıldığında harici komut dosyaları pratiktir.

JavaScript dosyaları .js dosya uzantısına sahiptir .

Harici bir komut dosyası kullanmak için komut dosyasının adını `src` bir etiketin (source) niteliğine girin `<script>`:

Örnek

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

Kendin dene "

Harici bir komut dosyası referansını istediğiniz gibi `<head>` veya içine yerleştirebilirsiniz . `<body>`

`<script>` Komut dosyası, tam olarak etiketin bulunduğu yerde bulunuyormuş gibi davranacaktır .

Harici komut dosyaları etiket içeremez `<script>`.

Harici JavaScript Avantajları

Komut dosyalarını harici dosyalara yerleştirmenin bazı avantajları vardır:

- HTML ve kodu ayırır
- HTML ve JavaScript'in okunmasını ve bakımını kolaylaştırır
- Önbelleğe alınmış JavaScript dosyaları sayfa yüklemelerini hızlandırabilir

Bir sayfaya birden fazla komut dosyası eklemek için birkaç komut dosyası etiketi kullanın:

Örnek

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Dış Referanslar

Harici bir komut dosyasına 3 farklı şekilde başvurulabilir:

- Tam URL ile (tam web adresi)
- Bir dosya yolu ile (/js/ gibi)
- Hiçbir yol olmadan

Bu örnekte myScript.js'ye bağlantı vermek için **tam URL kullanılır:**

Örnek

```
<script src="https://www.w3schools.com/js/myScript.js"></script>
```

Kendin dene "

Bu örnek, myScript.js'ye bağlanmak için bir **dosya yolu kullanır:**

Örnek

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

Kendin dene "

Bu örnekte myScript.js'ye bağlanmak için herhangi bir yol kullanılmaz:

Örnek

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

Kendin dene "

HTML Dosya Yolları bölümünde dosya yolları hakkında daha fazla bilgi edinebilirsiniz .

JavaScript Çıkışı

❮ Öncesi Sonraki ❯

JavaScript Görüntüleme Olanakları

JavaScript, verileri farklı şekillerde "görüntüleyebilir":

- . `innerHTML`
- Kullanarak HTML çıktısına yazma `document.write()` .
- Kullanarak bir uyarı kutusuna yazma `window.alert()` .
- Kullanarak tarayıcı konsoluna yazma `console.log()` .

innerHTML'yi kullanma

Bir HTML öğesine erişmek için JavaScript bu yöntemi kullanabilir `document.getElementById(id)` .

Özellik `id` HTML öğesini tanımlar. Özellik `innerHTML` HTML içeriğini tanımlar:

Örnek

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

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Kendin dene "

Bir HTML öğesinin innerHTML özelliğini değiştirmek, verileri HTML'de görüntülemenin yaygın bir yoludur.

document.write() işlevini kullanma

Test amacıyla aşağıdakilerin kullanılması uygundur `document.write()` :

Örnek

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

<h1>My First Web Page</h1>
```

```
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Kendin dene "

Bir HTML belgesi yüklendikten sonra document.write() işlevini kullanmak **mevcut tüm HTML'yi siler** :

Örnek

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Kendin dene "

document.write() yöntemi yalnızca test amacıyla kullanılmalıdır.

window.alert() işlevini kullanma

Verileri görüntülemek için bir uyarı kutusu kullanabilirsiniz:

Örnek

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

Kendin dene "

Anahtar kelimeyi atlayabilirsiniz `window`.

JavaScript'te pencere nesnesi genel kapsam nesnesidir. Bu, değişkenlerin, özelliklerin ve yöntemlerin varsayılan olarak pencere nesnesine ait olduğu anlamına gelir. Bu aynı zamanda anahtar kelimeyi belirtmenin isteğe bağlı olduğu anlamına da gelir `window`:

Örnek

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>

</body>
</html>
```

Kendin dene "

console.log()'u kullanma

`console.log()` Hata ayıklama amacıyla, verileri görüntülemek için tarayıcıdaki yöntemi çağırabilirsiniz .

Daha sonraki bir bölümde hata ayıklama hakkında daha fazla bilgi edineceksiniz.

Örnek

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

```
<script>
console.log(5 + 6);
</script>

</body>
</html>
```

Kendin dene "

JavaScript Yazdır

JavaScript'in herhangi bir yazdırma nesnesi veya yazdırma yöntemi yoktur.

Çıkış aygıtlarına JavaScript'ten erişemezsiniz.

`window.print()` Bunun tek istisnası, geçerli pencerenin içeriğini yazdırmak için tarayıcıdaki yöntemi çağırabilmenizdir .

Örnek

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

<button onclick="window.print()">Print this page</button>

</body>
</html>
```

JavaScript İfadeleri

❮ Öncesi Sonraki ❯

Örnek

```
let x, y, z;    // Statement 1
x = 5;          // Statement 2
y = 6;          // Statement 3
z = x + y;      // Statement 4
```

Kendin dene "

JavaScript Programları

Bir **bilgisayar programı,** bir bilgisayar tarafından "yürütülecek" "talimatların" bir listesidir.

Bir programlama dilinde bu programlama talimatlarına **ifadeler** denir .

Bir **JavaScript programı, programlama ifadelerinin** bir listesidir .

HTML'de JavaScript programları web tarayıcısı tarafından yürütülür.

JavaScript İfadeleri

JavaScript ifadeleri şunlardan oluşur:

Değerler, Operatörler, İfadeler, Anahtar Kelimeler ve Yorumlar.

Bu ifade, tarayıcıya "Merhaba Dolly" yazmasını söyler. id = "demo" içeren bir HTML öğesinin içinde:

Örnek

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Kendin dene "

Çoğu JavaScript programı birçok JavaScript ifadesi içerir.

İfadeler, yazıldıkları sırayla tek tek yürütülür.

JavaScript programlarına (ve JavaScript ifadelerine) genellikle JavaScript kodu denir.

Noktalı virgül;

Noktalı virgüller JavaScript ifadelerini ayırır.

Her yürütülebilir ifadenin sonuna noktalı virgül ekleyin:

Örnekler

```
let a, b, c;  // Declare 3 variables
a = 5;        // Assign the value 5 to a
b = 6;        // Assign the value 6 to b
c = a + b;    // Assign the sum of a and b to c
```

Kendin dene "

Noktalı virgülle ayrıldığında, bir satırda birden fazla ifadeye izin verilir:

```
a = 5; b = 6; c = a + b;
```

Kendin dene "

Web'de noktalı virgül içermeyen örnekler görebilirsiniz.
İfadelerin noktalı virgülle bitirilmesi gerekli değildir ancak önemle tavsiye edilir.

JavaScript Beyaz Boşluk

JavaScript birden fazla boşluğu yok sayar. Daha okunabilir hale getirmek için betiğinize beyaz boşluk ekleyebilirsiniz.

Aşağıdaki satırlar eşdeğerdir:

```
let person = "Hege";
let person="Hege";
```

Operatörlerin (= + - * /) etrafına boşluk koymak iyi bir uygulamadır:

```
let x = y + z;
```

JavaScript Satır Uzunluğu ve Satır Sonları

En iyi okunabilirlik için programcılar genellikle 80 karakterden uzun kod satırlarından kaçınmak isterler.

Bir JavaScript ifadesi tek bir satıra sığmıyorsa, onu bölmek için en iyi yer bir operatörün sonrasıdır:

Örnek

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Kendin dene "

JavaScript Kod Blokları

JavaScript ifadeleri küme parantezleri {...} içinde kod blokları halinde gruplandırılabilir.

Kod bloklarının amacı birlikte yürütülecek ifadeleri tanımlamaktır.

Bloklar halinde gruplandırılmış ifadeleri bulacağınız yerlerden biri de JavaScript işlevleridir:

Örnek

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
```

Kendin dene "

Bu derste kod blokları için 2 girinti alanı kullanıyoruz.
Bu eğitimin ilerleyen kısımlarında işlevler hakkında daha fazla bilgi edineceksiniz.

JavaScript Anahtar Kelimeleri

JavaScript ifadeleri, gerçekleştirilecek JavaScript eylemini tanımlamak için genellikle bir **anahtar kelimeyle** başlar .

Ayrılmış Kelimeler Referansımız tüm JavaScript anahtar kelimelerini listeler.

Bu eğitimde öğreneceğiniz bazı anahtar kelimelerin bir listesi:

Anahtar kelime	Tanım
var	Bir değişken bildirir
let	Bir blok değişkeni bildirir
const	Bir blok sabiti bildirir
if	Bir koşula göre yürütülecek ifade bloğunu işaretler
switch	Farklı durumlarda yürütülecek ifade bloğunu işaretler
for	Bir döngüde yürütülecek ifade bloğunu işaretler
function	Bir işlev bildirir
return	Bir fonksiyondan çıkar
try	Bir ifade bloğuna hata işlemeyi uygular

JavaScript anahtar kelimeleri ayrılmış kelimelerdir. Ayrılmış kelimeler değişkenlerin adı olarak kullanılamaz.

JavaScript Söz Dizimi

❮ Öncesi Sonraki ❯

JavaScript sözdizimi, JavaScript programlarının nasıl oluşturulduğunu gösteren kurallar kümesidir:

```
// How to create variables:
var x;
let y;

// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

JavaScript Değerleri

JavaScript sözdizimi iki tür değer tanımlar:

- Sabit değerler
- Değişken değerler

Sabit değerlere Literaller denir .

Değişken değerlerine Değişkenler denir .

JavaScript Değişmez Değerleri

Sabit değerler için en önemli iki sözdizimi kuralı şunlardır:

1. **Sayılar** ondalıklı veya ondalıksız yazılır:

```
10.50

1001
```

Kendin dene "

2. **Dizeler** çift veya tek tırnak içinde yazılan metinlerdir:

```
"John Doe"

'John Doe'
```

Kendin dene "

JavaScript Değişkenleri

Bir programlama dilinde **değişkenler** veri değerlerini **depolamak** için kullanılır .

`var` JavaScript , değişkenleri **bildirmek** için `let` ve anahtar sözcüklerini kullanır . `const`

Değişkenlere **değer atamak için eşittir işareti** kullanılır .

Bu örnekte x bir değişken olarak tanımlanmıştır. Daha sonra x'e 6 değeri atanır (verilir):

```
let x;
x = 6;
```

Kendin dene "

JavaScript Operatörleri

JavaScript, değerleri hesaplamak için **aritmetik operatörleri** ( ) kullanır : `+` `-` `*` `/`

```
(5 + 6) * 10
```

Kendin dene "

JavaScript, değişkenlere değer **atamak için bir atama operatörü** (`=`) kullanır :

```
let x, y;
x = 5;
y = 6;
```

Kendin dene "

JavaScript İfadeleri

İfade, bir değeri hesaplayan değerlerin, değişkenlerin ve operatörlerin birleşimidir.

Hesaplamaya değerlendirme denir.

Örneğin, 5 * 10, 50 olarak hesaplanır:

```
5 * 10
```

Kendin dene "

İfadeler ayrıca değişken değerler de içerebilir:

```
x * 10
```

Kendin dene "

Değerler sayılar ve dizeler gibi çeşitli türlerde olabilir.

Örneğin, "John" + " " + "Doe", "John Doe" olarak değerlendirilir:

```
"John" + " " + "Doe"
```

Kendin dene "

JavaScript Anahtar Kelimeleri

Gerçekleştirilecek eylemleri tanımlamak için JavaScript **anahtar kelimeleri kullanılır.**

Anahtar kelime `let`, tarayıcıya değişkenler oluşturmasını söyler:

```
let x, y;
x = 5 + 6;
y = x * 10;
```

Kendin dene "

Anahtar kelime `var` aynı zamanda tarayıcıya değişkenler oluşturmasını da söyler:

```
var x, y;
x = 5 + 6;
y = x * 10;
```

Kendin dene "

Bu örneklerde `var` veya kullanımı `let` aynı sonucu üretecektir.

Bu eğitimde `var` ve hakkında daha fazla bilgi edineceksiniz . `let`

JavaScript Yorumları

Tüm JavaScript ifadeleri "yürütülmez".

`//` Çift eğik çizgiden sonraki veya `/*` ve arasındaki kod **yorum** `*/` olarak değerlendirilir .

Yorumlar dikkate alınmaz ve yürütülmez:

```
let x = 5;   // I will be executed

// x = 6;   I will NOT be executed
```

Kendin dene "

Daha sonraki bir bölümde yorumlar hakkında daha fazla bilgi edineceksiniz.

JavaScript Tanımlayıcıları / Adları

Tanımlayıcılar JavaScript adlarıdır.

Tanımlayıcılar değişkenleri, anahtar sözcükleri ve işlevleri adlandırmak için kullanılır.

Yasal adlara ilişkin kurallar çoğu programlama dilinde aynıdır.

Bir JavaScript adı şununla başlamalıdır:

- Bir harf (AZ veya az)
- Dolar işareti ($)
- Veya alt çizgi (_)

Sonraki karakterler harf, rakam, alt çizgi veya dolar işareti olabilir.

Not

İsimlerde ilk karakter olarak sayılara izin verilmez.

Bu şekilde JavaScript tanımlayıcıları sayılardan kolayca ayırt edebilir.

JavaScript Büyük/Küçük Harfe Duyarlıdır

Tüm JavaScript tanımlayıcıları **büyük/küçük harfe duyarlıdır** .

Değişkenler `lastName` ve `lastname` , iki farklı değişkendir:

```
let lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

Kendin dene "

JavaScript, LET veya **Let'i let** anahtar sözcüğü olarak yorumlamaz .

JavaScript ve Camel Örneği

Geçmişte programcılar birden fazla kelimeyi tek bir değişken adı altında birleştirmenin farklı yollarını kullanmışlardır:

Kısa çizgiler:

ad, soyadı, ana kart, şehirlerarası.

JavaScript'te tirelere izin verilmez. Çıkarma işlemleri için ayrılmıştır.

Vurgulamak:

ad, soyadı, ana_kart, şehirlerarası.

Üst Camel Kasası (Pascal Kasası):

Ad, Soyadı, MasterCard, InterCity.

Alt Deve Kasası:

JavaScript programcıları küçük harfle başlayan deve tipini kullanma eğilimindedir:

ad, soyad, masterCard, interCity.

JavaScript Karakter Seti

JavaScript **Unicode** karakter kümesini kullanır.

Unicode dünyadaki (neredeyse) tüm karakterleri, noktalama işaretlerini ve simgeleri kapsar.

Daha yakından bakmak için lütfen Tam Unicode Referansımızı inceleyin .

JavaScript Yorumları

❮ Öncesi Sonraki ❯

JavaScript yorumları, JavaScript kodunu açıklamak ve onu daha okunaklı hale getirmek için kullanılabilir.

Alternatif kodu test ederken yürütmeyi önlemek için JavaScript yorumları da kullanılabilir.

Tek Satır Yorumlar

Tek satırlık yorumlar ile başlar `//`.

Satır sonu ile arasındaki metinler `//` JavaScript tarafından dikkate alınmayacaktır (yürütülmeyecektir).

Bu örnekte her kod satırından önce tek satırlık bir yorum kullanılır:

Örnek

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Kendin dene "

Bu örnekte, kodu açıklamak için her satırın sonunda tek satırlık bir yorum kullanılır:

Örnek

```
let x = 5;      // Declare x, give it the value of 5
let y = x + 2;  // Declare y, give it the value of x + 2
```

Kendin dene "

Çok Satırlı Yorumlar

Çok satırlı yorumlar ile başlar `/*` ve ile biter `*/`.

`/*` ile arasındaki herhangi bir metin `*/` JavaScript tarafından dikkate alınmayacaktır.

Bu örnekte, kodu açıklamak için çok satırlı bir yorum (yorum bloğu) kullanılır:

Örnek

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
```

```
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Kendin dene "

Tek satırlık yorumların kullanılması en yaygın olanıdır.
Blok yorumlar genellikle resmi dokümantasyon için kullanılır.

Yürütmeyi Önlemek İçin Yorumları Kullanma

Kodun yürütülmesini önlemek için yorumların kullanılması kod testi için uygundur.

`//` Bir kod satırının önüne eklemek, kod satırlarını çalıştırılabilir bir satırdan bir yoruma dönüştürür.

Bu örnekte, kod satırlarından birinin yürütülmesini önlemek için // kullanılır:

Örnek

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Kendin dene "

Bu örnek, birden fazla satırın yürütülmesini önlemek için bir yorum bloğu kullanır:

Örnek

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

JavaScript Değişkenleri

❮ Öncesi Sonraki ❯

Değişkenler Veri Depolamaya Yönelik Kaplardır

JavaScript Değişkenleri 4 şekilde bildirilebilir:

- Otomatik olarak
- Kullanma `var`
- Kullanma `let`
- Kullanma `const`

Bu ilk örnekte, `x`, `y` ve `z` bildirilmemiş değişkenlerdir.

İlk kullanıldıklarında otomatik olarak bildirilirler:

Örnek

```
x = 5;
y = 6;
z = x + y;
```

Kendin dene "

Not

Değişkenleri her zaman kullanımdan önce bildirmek iyi bir programlama uygulaması olarak kabul edilir.

Örneklerden tahmin edebilirsiniz:

- x, 5 değerini saklar
- y 6 değerini saklar
- z 11 değerini saklar

var kullanma örneği

```
var x = 5;
var y = 6;
var z = x + y;
```

Kendin dene "

Not

Anahtar `var` kelime 1995'ten 2015'e kadar tüm JavaScript kodlarında kullanıldı.

`let` ve anahtar kelimeleri `const` 2015 yılında JavaScript'e eklendi.

Anahtar `var` kelime yalnızca eski tarayıcılar için yazılan kodda kullanılmalıdır.

Let kullanma örneği

```
let x = 5;
let y = 6;
let z = x + y;
```

Kendin dene "

const kullanma örneği

```
const x = 5;
const y = 6;
const z = x + y;
```

Kendin dene "

Karma Örnek

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

Kendin dene "

İki değişken `price1` ve anahtar `price2` sözcüğüyle bildirilir `const`.

Bunlar sabit değerlerdir ve değiştirilemezler.

Değişken anahtar `total` kelimeyle bildirilir `let`.

Değer `total` değiştirilebilir.

var, let veya const ne zaman kullanılır?

1. Her zaman değişkenleri bildirin

`const` 2. Değerin değiştirilmemesi gerekiyorsa daima kullanın

`const` 3. Türün değiştirilmemesi gerekiyorsa daima kullanın (Diziler ve Nesneler)

4. Yalnızca `let` kullanamıyorsanız kullanın `const`

5. Yalnızca `var` eski tarayıcıları desteklemeniz ZORUNLU ise kullanın.

Tıpkı Cebir gibi

Tıpkı cebirde olduğu gibi değişkenler de değerleri tutar:

```
let x = 5;
let y = 6;
```

Tıpkı cebirde olduğu gibi değişkenler ifadelerde kullanılır:

```
let z = x + y;
```

Yukarıdaki örnekten toplamın 11 olarak hesaplandığını tahmin edebilirsiniz.

Not

Değişkenler değerlerin saklandığı kaplardır.

JavaScript Tanımlayıcıları

Tüm JavaScript **değişkenleri benzersiz** adlarla tanımlanmalıdır .

Bu benzersiz adlara tanımlayıcılar denir .

Tanımlayıcılar kısa adlar (x ve y gibi) veya daha açıklayıcı adlar (yaş, toplam, toplamHacim) olabilir.

Değişkenler (benzersiz tanımlayıcılar) için ad oluşturmaya ilişkin genel kurallar şunlardır:

- Adlar harf, rakam, alt çizgi ve dolar işareti içerebilir.
- İsimler bir harfle başlamalıdır.
- İsimler $ ve _ ile de başlayabilir (ancak bu derste bunu kullanmayacağız).
- Adlar büyük/küçük harfe duyarlıdır (y ve Y farklı değişkenlerdir).
- Ayrılmış kelimeler (JavaScript anahtar kelimeleri gibi) ad olarak kullanılamaz.

Not

JavaScript tanımlayıcıları büyük/küçük harfe duyarlıdır.

Atama Operatörü

JavaScript'te eşittir işareti (`=`) bir "eşittir" operatörü değil, bir "atama" operatörüdür.

Bu cebirden farklıdır. Cebirde aşağıdakilerin bir anlamı yoktur:

```
x = x + 5
```

Ancak JavaScript'te bu çok mantıklı: x + 5 değerini x'e atar.

(x + 5 değerini hesaplar ve sonucu x'e koyar. x'in değeri 5 artırılır.)

Not

"Eşittir" operatörü `==` JavaScript'teki gibi yazılmıştır.

JavaScript Veri Türleri

JavaScript değişkenleri 100 gibi sayıları ve "John Doe" gibi metin değerlerini tutabilir.

Programlamada metin değerlerine metin dizeleri denir.

JavaScript birçok veri türünü işleyebilir ancak şimdilik yalnızca sayıları ve dizeleri düşünün.

Dizeler çift veya tek tırnak içine yazılır. Sayılar tırnak işareti olmadan yazılır.

Bir sayıyı tırnak içine alırsanız, bu bir metin dizesi olarak değerlendirilir.

Örnek

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Kendin dene "

JavaScript Değişkeni Bildirmek

JavaScript'te bir değişken oluşturmaya değişkenin "bildirilmesi" denir.

`var` or anahtar sözcüğüyle bir JavaScript değişkeni bildirirsiniz `let` :

```
var carName;
```

veya:

```
let carName;
```

Bildirimden sonra değişkenin hiçbir değeri yoktur (teknik olarak öyledir `undefined`).

Değişkene bir değer **atamak** için eşittir işaretini kullanın:

```
carName = "Volvo";
```

Değişkeni bildirdiğinizde ona bir değer de atayabilirsiniz:

```
let carName = "Volvo";
```

Aşağıdaki örnekte adında bir değişken oluşturup `carName` ona "Volvo" değerini atadık.

Daha sonra, bir HTML paragrafının içindeki değeri id = "demo" ile "çıkartıyoruz":

Örnek

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

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Kendin dene "

Not

Tüm değişkenleri bir betiğin başında bildirmek iyi bir programlama uygulamasıdır.

Tek İfade, Birçok Değişken

Tek bir ifadede birçok değişkeni tanımlayabilirsiniz.

İfadeyi şununla başlatın `let` ve değişkenleri virgülle **ayırın** :

Örnek

```
let person = "John Doe", carName = "Volvo", price = 200;
```

Kendin dene "

Bir bildirim birden fazla satıra yayılabilir:

Örnek

```
let person = "John Doe",
carName = "Volvo",
price = 200;
```

Kendin dene "

Değer = tanımsız

Bilgisayar programlarında değişkenler genellikle değer olmadan bildirilir. Değer, hesaplanması gereken bir şey veya kullanıcı girişi gibi daha sonra sağlanacak bir şey olabilir.

Değer olmadan bildirilen bir değişken, değere sahip olacaktır `undefined`.

carName değişkeni `undefined` bu ifadenin yürütülmesinden sonra şu değere sahip olacaktır:

Örnek

```
let carName;
```

Kendin dene "

JavaScript Değişkenlerini Yeniden Bildirmek

İle bildirilen bir JavaScript değişkenini yeniden bildirirseniz `var` değerini kaybetmez.

`carName` Bu ifadelerin yürütülmesinden sonra değişken hala "Volvo" değerine sahip olacaktır:

Örnek

```
var carName = "Volvo";
var carName;
```

Kendin dene "

Not

`let` veya ile bildirilen bir değişkeni yeniden bildiremezsiniz `const`.

Bu işe yaramayacak:

```
let carName = "Volvo";
let carName;
```

JavaScript Aritmetiği

`=` Cebirde olduğu gibi, ve gibi operatörleri kullanarak JavaScript değişkenleriyle aritmetik yapabilirsiniz `+`:

Örnek

```
let x = 5 + 2 + 3;
```

Kendin dene "

Ayrıca dizeler de ekleyebilirsiniz, ancak dizeler birleştirilecektir:

Örnek

```
let x = "John" + " " + "Doe";
```

Kendin dene "

Ayrıca şunu deneyin:

Örnek

```
let x = "5" + 2 + 3;
```

Kendin dene "

Not

Bir sayıyı tırnak içine alırsanız, sayıların geri kalanı dize olarak kabul edilir ve birleştirilir.

Şimdi şunu deneyin:

Örnek

```
let x = 2 + 3 + "5";
```

Kendin dene "

JavaScript Dolar İşareti $

JavaScript dolar işaretini bir harf gibi ele aldığından, $ içeren tanımlayıcılar geçerli değişken adlarıdır:

Örnek

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
```

Kendin dene "

Dolar işaretini kullanmak JavaScript'te pek yaygın değildir, ancak profesyonel programcılar bunu genellikle bir JavaScript kitaplığındaki ana işlev için takma ad olarak kullanırlar.

Örneğin, JavaScript kitaplığı jQuery'de, `$` HTML öğelerini seçmek için ana işlev kullanılır. JQuery'de `$("p");` "tüm p öğelerini seç" anlamına gelir.

JavaScript Alt Çizgisi (_)

JavaScript alt çizgiyi bir harf gibi ele aldığından, _ içeren tanımlayıcılar geçerli değişken adlarıdır:

Örnek

```
let _lastName = "Johnson";
let _x = 2;
let _100 = 5;
```

Kendin dene "

Alt çizginin kullanılması JavaScript'te pek yaygın değildir, ancak profesyonel programcılar arasındaki bir gelenek, bunu "özel (gizli)" değişkenler için takma ad olarak kullanmaktır.

Egzersizlerle Kendinizi Test Edin

Egzersiz yapmak:

Adlı bir değişken oluşturun `carName` ve değeri `Volvo` ona atayın.

```
izin vermek        = "       ";
```

Cevabı gönder "

Egzersizi Başlat

JavaScript İzin Ver

❮ Öncesi Sonraki ❯

Anahtar kelime ES6'da (2015) `let` tanıtıldı

Blok Kapsamına `let` sahip olarak bildirilen değişkenler

İle bildirilen değişkenler **kullanımdan** önce `let` bildirilmelidir

İle bildirilen değişkenler aynı kapsamda **yeniden** `let` bildirilemez

Blok Kapsamı

ES6'dan (2015) önce, JavaScript'te Block Scope yoktu .

JavaScript'in Global Kapsamı ve **İşlev Kapsamı** vardı .

ES6 iki yeni JavaScript anahtar kelimesini tanıttı: `let` ve `const` .

Bu iki anahtar kelime JavaScript'te **Blok Kapsamı sağladı:**

Örnek

{ } bloğunun içinde bildirilen değişkenlere bloğun dışından erişilemez:

```
{
  let x = 2;
}
// x can NOT be used here
```

Küresel Kapsam

Always ile bildirilen değişkenler **Global Scope'a** `var` sahiptir .

Anahtar kelimeyle bildirilen değişkenler `var` blok kapsamına sahip OLAMAZ:

Örnek

{ } bloğunun içinde bildirilen değişkenlere `var` bloğun dışından erişilebilir:

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

Yeniden beyan edilemez

İle tanımlanan değişkenler yeniden `let` **bildirilemez** .

İle bildirilen bir değişkeni yanlışlıkla yeniden bildiremezsiniz `let` .

Bunu seninle `let` yapamazsınız :

```
let x = "John Doe";
```

```
let x = 0;
```

İle tanımlanan değişkenler yeniden `var` **bildirilebilir** .

Bunu kullanarak **şunları** `var` yapabilirsiniz :

```
var x = "John Doe";

var x = 0;
```

Değişkenleri Yeniden Bildirmek

Anahtar kelimeyi kullanarak bir değişkeni yeniden bildirmek `var` sorunlara neden olabilir.

Bir bloğun içindeki bir değişkeni yeniden bildirmek, bloğun dışındaki değişkeni de yeniden bildirecektir:

Örnek

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

Kendin dene "

Anahtar kelimeyi kullanarak bir değişkeni yeniden bildirmek `let` bu sorunu çözebilir.

Bir bloğun içindeki bir değişkeni yeniden bildirmek, bloğun dışındaki değişkeni yeniden bildirmez:

Örnek

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

Kendin dene "

var, let ve const arasındaki fark

	Kapsam	Yeniden beyan et	Yeniden ata	Kaldırılmış	Bunu bağlar
var	HAYIR	Evet	Evet	Evet	Evet
izin vermek	Evet	HAYIR	Evet	HAYIR	HAYIR
yapı	Evet	HAYIR	HAYIR	HAYIR	HAYIR

Ne iyi?

`let` ve **blok kapsamına** `const` sahip .

`let` ve **yeniden beyan** `const` edilemez .

`let` ve kullanımdan önce **beyan** `const` edilmelidir .

`let` ve `const` bağlamaz . `this`

`let` ve `const` kaldırılmaz .

İyi olmayan ne?

`var` beyan edilmesine gerek yoktur.

`var` kaldırılır.

`var` buna bağlanır.

Tarayıcı Desteği

`let` ve anahtar sözcükleri `const` Internet Explorer 11 veya daha önceki sürümlerde desteklenmez.

Aşağıdaki tabloda tam destekli ilk tarayıcı sürümleri tanımlanmaktadır:

				
Chrome 49	Edge 12	Firefox 36	Safari 11	Opera 36
Mar, 2016	Jul, 2015	Jan, 2015	Sep, 2017	Mar, 2016

Yeniden beyan ediliyor

Bir JavaScript değişkeninin yeniden bildirilmesine `var` programın herhangi bir yerinde izin verilir:

Örnek

```
var x = 2;
// Now x is 2

var x = 3;
// Now x is 3
```

Kendin dene "

ile `let` aynı blokta bir değişkenin yeniden bildirilmesine izin VERİLMEZ:

Örnek

```
var x = 2;   // Allowed
let x = 3;   // Not allowed

{
let x = 2;   // Allowed
let x = 3;   // Not allowed
}

{
let x = 2;   // Allowed
var x = 3;   // Not allowed
}
```

Bir değişkenin `let` başka bir blokta , ile yeniden bildirilmesine izin verilir:

Örnek

```
let x = 2;   // Allowed

{
let x = 3;   // Allowed
}

{
let x = 4;    // Allowed
}
```

Kendin dene "

Bırakın Kaldırma

İle tanımlanan değişkenler en **üste çıkarılır** `var` ve herhangi bir zamanda başlatılabilir.

Anlamı: Değişkeni bildirilmeden önce kullanabilirsiniz:

Örnek

Tamamdır:

```
carName = "Volvo";
var carName;
```

Kendin dene "

Kaldırma hakkında daha fazla bilgi edinmek istiyorsanız JavaScript Kaldırma bölümünü inceleyin .

İle tanımlanan değişkenler `let` de bloğun tepesine kaldırılır ancak başlatılmaz.

Anlamı: Bir `let` değişkenin bildirilmeden önce kullanılması şununla sonuçlanacaktır `ReferenceError` :

Örnek

```
carName = "Saab";
let carName = "Volvo";
```

JavaScript Yapısı

❮ Öncesi Sonraki ❯

Anahtar kelime ES6'da (2015) `const` tanıtıldı

İle tanımlanan değişkenler **Yeniden** `const` Bildirilemez

İle tanımlanan değişkenler **Yeniden** `const` Atanamaz

Blok Kapsamına `const` sahip ile tanımlanan değişkenler

Yeniden Atanamaz

Anahtar kelimeyle tanımlanan bir değişken `const` yeniden atanamaz:

Örnek

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

Kendin dene "

Atanmalı

JavaScript `const` değişkenlerine bildirildiklerinde bir değer atanmalıdır:

Doğru

```
const PI = 3.14159265359;
```

Yanlış

```
const PI;
PI = 3.14159265359;
```

JavaScript const ne zaman kullanılır?

`const` **Değerin değiştirilmemesi gerektiğini bildiğinizde her zaman bir değişkeni bildirin .**

`const` Bildirdiğinizde kullanın :

- Yeni bir Dizi
- Yeni bir Nesne
- Yeni bir İşlev
- Yeni bir RegExp

Sabit Nesneler ve Diziler

Anahtar kelime `const` biraz yanıltıcıdır.

Sabit bir değer tanımlamaz. Bir değere sabit bir referans tanımlar.

Bu nedenle şunları yapamazsınız:

- Sabit bir değeri yeniden atayın
- Sabit bir diziyi yeniden atama
- Sabit bir nesneyi yeniden atama

Ama sen yapabilirsin:

- Sabit dizinin elemanlarını değiştirme
- Sabit nesnenin özelliklerini değiştirme

Sabit Diziler

Sabit bir dizinin elemanlarını değiştirebilirsiniz:

Örnek

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

Kendin dene "

Ancak diziyi yeniden atayamazsınız:

Örnek

```
const cars = ["Saab", "Volvo", "BMW"];

cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

Kendin dene "

Sabit Nesneler

Sabit bir nesnenin özelliklerini değiştirebilirsiniz:

Örnek

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};

// You can change a property:
car.color = "red";

// You can add a property:
car.owner = "Johnson";
```

Kendin dene "

Ancak nesneyi yeniden atayamazsınız:

Örnek

```
const car = {type:"Fiat", model:"500", color:"white"};

car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR
```

Kendin dene "

var, let ve const arasındaki fark

	Kapsam	Yeniden beyan et	Yeniden ata	Kaldırılmış	Bunu bağlar
var	HAYIR	Evet	Evet	Evet	Evet
izin vermek	Evet	HAYIR	Evet	HAYIR	HAYIR
yapı	Evet	HAYIR	HAYIR	HAYIR	HAYIR

Ne iyi?

`let` ve **blok kapsamına** `const` sahip .

`let` ve **yeniden beyan** `const` edilemez .

`let` ve kullanımdan önce **beyan** `const` edilmelidir .

`let` ve `const` bağlamaz . `this`

`let` ve `const` kaldırılmaz .

İyi olmayan ne?

`var` beyan edilmesine gerek yoktur.

`var` kaldırılır.

`var` buna bağlanır.

Tarayıcı Desteği

`let` ve anahtar sözcükleri `const` Internet Explorer 11 veya daha önceki sürümlerde desteklenmez.

Aşağıdaki tabloda tam destekli ilk tarayıcı sürümleri tanımlanmaktadır:

				
Chrome 49	Edge 12	Firefox 36	Safari 11	Opera 36
Mar, 2016	Jul, 2015	Jan, 2015	Sep, 2017	Mar, 2016

Blok Kapsamı

Bir değişkeni ile bildirmek, **Block** Scope'a `const` benzer . `let`

Bu örnekte blokta bildirilen x, bloğun dışında bildirilen x ile aynı değildir:

Örnek

```
const x = 10;
// Here x is 10

{
const x = 2;
// Here x is 2
}

// Here x is 10
```

Kendin dene "

Blok kapsamı hakkında daha fazla bilgiyi JavaScript Kapsamı bölümünde bulabilirsiniz .

Yeniden beyan ediliyor

`var` Bir programın herhangi bir yerinde bir JavaScript değişkeninin yeniden bildirilmesine izin verilir:

Örnek

```
var x = 2;      // Allowed
var x = 3;      // Allowed
x = 4;          // Allowed
```

Aynı kapsamda mevcut bir değişkenin `var` veya `let` değişkenin yeniden bildirilmesine izin verilmez: `const`

Örnek

```
var x = 2;      // Allowed
const x = 2;    // Not allowed

{
let x = 2;      // Allowed
const x = 2;    // Not allowed
}

{
const x = 2;    // Allowed
const x = 2;    // Not allowed
}
```

`const` Aynı kapsamda mevcut bir değişkenin yeniden atanmasına izin verilmez:

Örnek

```
const x = 2;      // Allowed
x = 2;            // Not allowed
var x = 2;        // Not allowed
```

```
let x = 2;        // Not allowed
const x = 2;      // Not allowed

{
  const x = 2;    // Allowed
  x = 2;          // Not allowed
  var x = 2;      // Not allowed
  let x = 2;      // Not allowed
  const x = 2;    // Not allowed
}
```

Bir değişkenin `const` başka bir kapsamda veya başka bir blokta yeniden bildirilmesine izin verilir:

Örnek

```
const x = 2;       // Allowed

{
  const x = 3;   // Allowed
}

{
  const x = 4;   // Allowed
}
```

Kaldırma

İle tanımlanan değişkenler en **üste çıkarılır** `var` ve herhangi bir zamanda başlatılabilir.

Anlamı: Değişkeni bildirilmeden önce kullanabilirsiniz:

Örnek

Tamamdır:

```
carName = "Volvo";
var carName;
```

Kendin dene "

Kaldırma hakkında daha fazla bilgi edinmek istiyorsanız JavaScript Kaldırma bölümünü inceleyin .

İle tanımlanan değişkenler `const` de en üste çıkarılır ancak başlatılmaz.

Anlamı: Bir `const` değişkenin bildirilmeden önce kullanılması şununla sonuçlanacaktır `ReferenceError` :

Örnek

```
alert (carName);
const carName = "Volvo";
```

JavaScript Operatörleri

❮ Öncesi Sonraki ❯

Javascript operatörleri farklı türde matematiksel ve mantıksal hesaplamaları gerçekleştirmek için kullanılır.

Örnekler:

Atama **Operatörü** = değerleri atar

Toplama Operatörü + **değerleri** ekler

Çarpma Operatörü * **değerleri** çarpar

Karşılaştırma **Operatörü** > değerleri karşılaştırır

JavaScript Ataması

Atama **Operatörü** (`=`) bir değişkene bir değer atar:

Ödev Örnekleri

```
let x = 10;
```

Kendin dene "

```
// Assign the value 5 to x
let x = 5;
// Assign the value 2 to y
let y = 2;
// Assign the value x + y to z:
let z = x + y;
```

Kendin dene "

JavaScript Eklemesi

Toplama Operatörü ( ) `+` sayıları ekler:

Ekleme

```
let x = 5;
let y = 2;
let z = x + y;
```

Kendin dene "

JavaScript Çarpma

Çarpma **Operatörü** (`*`) sayıları çarpar:

Çarpma

```
let x = 5;
let y = 2;
let z = x * y;
```

Kendin dene "

JavaScript Operatörlerinin Türleri

Farklı türde JavaScript operatörleri vardır:

- Aritmetik operatörler
- Atama Operatörleri
- Karşılaştırma Operatörleri
- Dize Operatörleri
- Mantıksal operatörler
- Bitsel Operatörler
- Üçlü Operatörler
- Tip Operatörler

JavaScript Aritmetik Operatörleri

Aritmetik Operatörler sayılar üzerinde aritmetik işlem yapmak için kullanılır:

Aritmetik Operatörler Örneği

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

Kendin dene "

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Not

Aritmetik operatörler JS Aritmetik bölümünde tam olarak açıklanmıştır .

JavaScript Atama Operatörleri

Atama operatörleri, JavaScript değişkenlerine değer atar.

Toplama **Atama Operatörü** (`+=`), bir değişkene değer ekler.

Atama

```
let x = 10;
x += 5;
```

Kendin dene "

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Not

Atama operatörleri JS Atama bölümünde tam olarak açıklanmıştır .

JavaScript Karşılaştırma Operatörleri

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

?	ternary operator

Not

Karşılaştırma operatörleri JS Karşılaştırmaları bölümünde tam olarak açıklanmıştır .

JavaScript Dize Karşılaştırması

Yukarıdaki karşılaştırma operatörlerinin tümü dizelerde de kullanılabilir:

Örnek

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
```

Kendin dene "

Dizelerin alfabetik olarak karşılaştırıldığını unutmayın:

Örnek

```
let text1 = "20";
let text2 = "5";
let result = text1 < text2;
```

Kendin dene "

JavaScript Dizesi Ekleme

`+` Ayrıca dizeleri eklemek (birleştirmek) için de kullanılabilir :

Örnek

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

Kendin dene "

Atama operatörü `+=` aynı zamanda dizeleri eklemek (birleştirmek) için de kullanılabilir:

Örnek

```
let text1 = "What a very ";
text1 += "nice day";
```

Text1'in sonucu şöyle olacaktır:

```
What a very nice day
```

Kendin dene "

Not

+ operatörü dizelerde kullanıldığında birleştirme operatörü olarak adlandırılır.

Dizeler ve Sayılar Ekleme

İki sayı eklemek toplamı döndürür, ancak bir sayı ve bir dize eklemek bir dize döndürür:

Örnek

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

x , *y* ve *z'nin* sonucu şöyle olacaktır:

```
10
55
Hello5
```

Kendin dene "

Not

Bir sayı ve bir dize eklerseniz sonuç bir dize olacaktır!

JavaScript Mantıksal Operatörleri

Operator	Description
&&	logical and
||	logical or
!	logical not

Not

Mantıksal operatörler JS Karşılaştırmaları bölümünde tam olarak açıklanmıştır .

JavaScript Türü Operatörleri

Operator	Description
typeof	Returns the type of a variable

instanceof	Returns true if an object is an instance of an object type

Not

Tür operatörleri JS Tür Dönüştürme bölümünde tam olarak açıklanmıştır .

JavaScript Bitsel Operatörler

Bit operatörleri 32 bitlik sayılar üzerinde çalışır.

İşlemdeki herhangi bir sayısal işlenen 32 bitlik bir sayıya dönüştürülür. Sonuç tekrar bir JavaScript numarasına dönüştürülür.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

Yukarıdaki örneklerde 4 bitlik imzasız örnekler kullanılmaktadır. Ancak JavaScript 32 bit imzalı sayılar kullanır.
Bu nedenle JavaScript'te ~5 değeri 10 değerini döndürmez. -6 değerini döndürür.
~00000000000000000000000000000101 11111111111111111111111111111010 değerini döndürecek

Bitsel operatörler JS Bitwise bölümünde tam olarak açıklanmıştır .

Egzersizlerle Kendinizi Test Edin

Egzersiz yapmak:

`10` ile çarpın `5` ve sonucu uyarın.

```
uyarı(10   5);
```

Cevabı gönder "

Egzersizi Başlat

JavaScript Aritmetiği

❮ Öncesi Sonraki ❯

JavaScript Aritmetik Operatörleri

Aritmetik operatörler sayılar (değişkenler veya değişkenler) üzerinde aritmetik işlem gerçekleştirir.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Aritmetik işlemler

Tipik bir aritmetik işlem iki sayı üzerinde çalışır.

İki sayı değişmez olabilir:

Örnek

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

Kendin dene "

veya değişkenler:

Örnek

```
let x = a + b;
```

Kendin dene "

veya ifadeler:

Örnek

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

Kendin dene "

Operatörler ve İşlenenler

Sayılara (bir aritmetik işlemde) **işlenenler** denir .

İşlem (iki işlenen arasında gerçekleştirilecek) bir **operatör** tarafından tanımlanır .

İşlenen	Şebeke	İşlenen
100	+	50

Ekleme

Toplama operatörü (`+`) sayıları toplar:

Örnek

```
let x = 5;
let y = 2;
let z = x + y;
```

Kendin dene "

Çıkarma

Çıkarma **operatörü** (`-`) sayıları çıkarır.

Örnek

```
let x = 5;
let y = 2;
let z = x - y;
```

Kendin dene "

Çarpma

Çarpma operatörü (`*`) sayıları çarpar.

Örnek

```
let x = 5;
let y = 2;
let z = x * y;
```

Kendin dene "

Bölme

Bölme operatörü (`/`) sayıları böler.

Örnek

```
let x = 5;
let y = 2;
let z = x / y;
```

Kendin dene "

Kalan

Modül operatörü (`%`), bölme kalanını döndürür.

Örnek

```
let x = 5;
let y = 2;
let z = x % y;
```

Kendin dene "

Aritmetikte iki tam sayının bölünmesi bir **bölüm** ve bir **kalan** üretir .

Matematikte modulo işleminin sonucu, aritmetik bölmenin **kalanıdır** .

Artırma

Arttırma operatörü (`++`) sayıları artırır.

Örnek

```
let x = 5;
x++;
let z = x;
```

Kendin dene "

Azalan

Azaltma operatörü (**)** `--` sayıları azaltır.

Örnek

```
let x = 5;
x--;
let z = x;
```

Kendin dene "

Üs alma

Üs alma operatörü (`**`), birinci işleneni ikinci işlenenin üssüne yükseltir.

Örnek

```
let x = 5;
let z = x ** 2;
```

Kendin dene "

x ** y aşağıdakiyle aynı sonucu üretir `Math.pow(x,y)` :

Örnek

```
let x = 5;
let z = Math.pow(x,2);
```

Kendin dene "

Operatör Önceliği

Operatör önceliği, bir aritmetik ifadede işlemlerin gerçekleştirilme sırasını açıklar.

Örnek

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

Kendin dene "

Yukarıdaki örneğin sonucu 150*3 ile aynı mı, yoksa 100 + 150 ile aynı mı?

Önce toplama mı yoksa çarpma mı yapılır?

Geleneksel okul matematiğinde olduğu gibi önce çarpma işlemi yapılır.

Çarpma (`*`) ve bölme ( ), toplama ( ) ve çıkarma ( ) işlemlerinden daha yüksek **önceliğe** `/` sahiptir . `+` `-`

Ve (okul matematiğinde olduğu gibi) öncelik parantez kullanılarak değiştirilebilir.

Parantez kullanıldığında ilk önce parantez içindeki işlemler hesaplanır:

Örnek

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

Kendin dene "

Birçok işlem aynı önceliğe sahip olduğunda (toplama ve çıkarma veya çarpma ve bölme gibi), soldan sağa doğru hesaplanır:

Örnekler

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

Kendin dene "

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

Kendin dene "

Not

Operatör öncelik değerlerinin tam listesi için şu adrese gidin:

JavaScript Operatör Öncelik Değerleri .

Egzersizlerle Kendinizi Test Edin

Egzersiz yapmak:

`10` Şuna bölün `2` ve sonucu uyarın.

```
uyarı(10  2);
```

Cevabı gönder "

Egzersizi Başlat

JavaScript Assignment

‹ Previous Next ›

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Shift Assignment Operators

Operator	Example	Same As
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

Bitwise Assignment Operators

Operator	Example	Same As
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

Logical Assignment Operators

Operator	Example	Same As
&&=	x &&= y	x = x && (x = y)
||=	x ||= y	x = x || (x = y)
??=	x ??= y	x = x ?? (x = y)

Note

The Logical assignment operators are ES2020.

The = Operator

The **Simple Assignment Operator** assigns a value to a variable.

Simple Assignment Examples

```
let x = 10;
```

Try it Yourself »

```
let x = 10 + y;
```

Try it Yourself »

The += Operator

The **Addition Assignment Operator** adds a value to a variable.

Addition Assignment Examples

```
let x = 10;
x += 5;
```

Try it Yourself »

```
let text = "Hello"; text += " World";
```

Try it Yourself »

The -= Operator

The **Subtraction Assignment Operator** subtracts a value from a variable.

Subtraction Assignment Example

```
let x = 10;
x -= 5;
```

Try it Yourself »

The *= Operator

The **Multiplication Assignment Operator** multiplies a variable.

Multiplication Assignment Example

```
let x = 10;
x *= 5;
```

Try it Yourself »

The **= Operator

The **Exponentiation Assignment Operator** raises a variable to the power of the operand.

Exponentiation Assignment Example

```
let x = 10;
x **= 5;
```

Try it Yourself »

The /= Operator

The **Division Assignment Operator** divides a variable.

Division Assignment Example

```
let x = 10;
x /= 5;
```

Try it Yourself »

The %= Operator

The **Remainder Assignment Operator** assigns a remainder to a variable.

Remainder Assignment Example

```
let x = 10;
x %= 5;
```

Try it Yourself »

The <<= Operator

The **Left Shift Assignment Operator** left shifts a variable.

Left Shift Assignment Example

```
let x = -100;
x <<= 5;
```

Try it Yourself »

The >>= Operator

The **Right Shift Assignment Operator** right shifts a variable (signed).

Right Shift Assignment Example

```
let x = -100;
x >>= 5;
```

Try it Yourself »

The >>>= Operator

The **Unsigned Right Shift Assignment Operator** right shifts a variable (unsigned).

Unsigned Right Shift Assignment Example

```
let x = -100;
x >>>= 5;
```

Try it Yourself »

The &= Operator

The **Bitwise AND Assignment Operator** does a bitwise AND operation on two operands and assigns the result to the the variable.

Bitwise AND Assignment Example

```
let x = 10;
x &= 5;
```

Try it Yourself »

The |= Operator

The **Bitwise OR Assignment Operator** does a bitwise OR operation on two operands and assigns the result to the variable.

Bitwise OR Assignment Example

```
let x = 10;
x |= 5;
```

Try it Yourself »

The ^= Operator

The **Bitwise XOR Assignment Operator** does a bitwise XOR operation on two operands and assigns the result to the variable.

Bitwise XOR Assignment Example

```
let x = 10;
x ^= 5;
```

Try it Yourself »

The &&= Operator

The **Logical AND assignment operator** is used between two values.

If the first value is true, the second value is assigned.

Logical AND Assignment Example

```
let x = 10;
x &&= 5;
```

Try it Yourself »

The `&&=` operator is an ES2020 feature.

The ||= Operator

The **Logical OR assignment operator** is used between two values.

If the first value is false, the second value is assigned.

Logical OR Assignment Example

```
let x = 10;
x ||= 5;
```

Try it Yourself »

The `||=` operator is an ES2020 feature.

The ??= Operator

The **Nullish coalescing assignment operator** is used between two values.

If the first value is undefined or null, the second value is assigned.

Nullish Coalescing Assignment Example

```
let x;
x ??= 5;
```

Try it Yourself »

The `??=` operator is an ES2020 feature.

Test Yourself With Exercises

Exercise:

Use the correct **assignment operator** that will result in `x` being `15` (same as `x = x + y`).

```
x = 10;
y = 5;
x    y;
```

Submit Answer »

Start the Exercise

JavaScript Veri Türleri

❮ Öncesi Sonraki ❯

JavaScript'in 8 Veri Türü Vardır

1. String
2. Sayı
3. Bigint
4. Boolean
5. Tanımsız
6. Boş
7. Sembol
8. Nesne

Nesne Veri Türü

Nesne veri türü şunları içerebilir:

1. Bir nesne
2. Bir dizi
3. Bir tarih

Örnekler

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

Not

Bir JavaScript değişkeni her türlü veriyi tutabilir.

Veri Türleri Kavramı

Programlamada veri türleri önemli bir kavramdır.

Değişkenler üzerinde işlem yapabilmek için tür hakkında bir şeyler bilmek önemlidir.

Veri türleri olmadan bir bilgisayar bunu güvenli bir şekilde çözemez:

```
let x = 16 + "Volvo";
```

On altıya "Volvo"yu eklemenin bir anlamı var mı? Hata mı üretecek yoksa sonuç mu üretecek?

JavaScript yukarıdaki örneği şu şekilde ele alacaktır:

```
let x = "16" + "Volvo";
```

Not

Bir sayı ve bir dize eklerken, JavaScript bu sayıyı bir dize olarak ele alır.

Örnek

```
let x = 16 + "Volvo";
```

Kendin dene "

Örnek

```
let x = "Volvo" + 16;
```

Kendin dene "

JavaScript ifadeleri soldan sağa doğru değerlendirir. Farklı diziler farklı sonuçlar doğurabilir:

JavaScript:

```
let x = 16 + 4 + "Volvo";
```

Sonuç:

```
20Volvo
```

Kendin dene "

JavaScript:

```
let x = "Volvo" + 16 + 4;
```

Sonuç:

```
Volvo164
```

Kendin dene "

İlk örnekte, JavaScript 16 ve 4'ü "Volvo"ya ulaşana kadar sayı olarak ele alır.

İkinci örnekte, ilk işlenen bir dize olduğundan, tüm işlenenler dize olarak ele alınır.

JavaScript Türleri Dinamiktir

JavaScript'in dinamik türleri vardır. Bu, aynı değişkenin farklı veri türlerini tutmak için kullanılabileceği anlamına gelir:

Örnek

```
let x;       // Now x is undefined
x = 5;       // Now x is a Number
x = "John";  // Now x is a String
```

Kendin dene "

JavaScript Dizeleri

Bir dize (veya bir metin dizesi), "John Doe" gibi bir karakter dizisidir.

Dizeler tırnak işaretleri ile yazılır. Tek veya çift tırnak kullanabilirsiniz:

Örnek

```
// Using double quotes:
let carName1 = "Volvo XC60";

// Using single quotes:
let carName2 = 'Volvo XC60';
```

Kendin dene "

Bir dizenin içindeki tırnak işaretlerini, dizeyi çevreleyen tırnak işaretleri ile eşleşmedikleri sürece kullanabilirsiniz:

Örnek

```
// Single quote inside double quotes:
let answer1 = "It's alright";

// Single quotes inside double quotes:
let answer2 = "He is called 'Johnny'";

// Double quotes inside single quotes:
let answer3 = 'He is called "Johnny"';
```

Kendin dene "

Bu eğitimin ilerleyen kısımlarında **dizeler** hakkında daha fazla bilgi edineceksiniz .

JavaScript Numaraları

Tüm JavaScript sayıları ondalık sayılar (kayan nokta) olarak saklanır.

Sayılar ondalık sayılarla veya ondalık sayı olmadan yazılabilir:

Örnek

```
// With decimals:
let x1 = 34.00;

// Without decimals:
let x2 = 34;
```

Kendin dene "

Üstel Gösterim

Ekstra büyük veya ekstra küçük sayılar bilimsel (üstel) gösterimle yazılabilir:

Örnek

```
let y = 123e5;    // 12300000
let z = 123e-5;   // 0.00123
```

Kendin dene "

Not

Çoğu programlama dilinde birçok sayı türü bulunur:

Tam sayılar (tamsayılar):
bayt (8 bit), kısa (16 bit), int (32 bit), uzun (64 bit)

Gerçek sayılar (kayan nokta):
float (32 bit), double (64 bit).

Javascript numaraları her zaman tek tiptir:
double (64-bit kayan nokta).

Bu eğitimin ilerleyen kısımlarında **sayılar** hakkında daha fazla bilgi edineceksiniz .

JavaScript BigInt

Tüm JavaScript numaraları 64 bit kayan nokta formatında saklanır.

JavaScript BigInt , normal bir JavaScript Numarasıyla temsil edilemeyecek kadar büyük tamsayı değerlerini depolamak için kullanılabilen yeni bir veri türüdür (ES2020).

Örnek

```
let x = BigInt("12345678901234567890123456789O");
```

Kendin dene "

Bu eğitimin ilerleyen kısımlarında **BigInt** hakkında daha fazla bilgi edineceksiniz .

JavaScript Boole'ları

Boolean'lar yalnızca iki değere sahip olabilir: `true` veya `false` .

Örnek

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Kendin dene "

Boolean'lar genellikle koşullu testlerde kullanılır.

Bu eğitimin ilerleyen kısımlarında **boolean'lar** hakkında daha fazla bilgi edineceksiniz .

JavaScript Dizileri

JavaScript dizileri köşeli parantezlerle yazılır.

Dizi öğeleri virgülle ayrılır.

Aşağıdaki kod, `cars` üç öğe (araba adı) içeren, adında bir dizi bildirir (oluşturur):

Örnek

```
const cars = ["Saab", "Volvo", "BMW"];
```

Kendin dene "

Dizi dizinleri sıfır tabanlıdır; bu, ilk öğenin [0], ikincisinin [1] vb. olduğu anlamına gelir.

Bu eğitimin ilerleyen kısımlarında **diziler** hakkında daha fazla bilgi edineceksiniz .

JavaScript Nesneleri

JavaScript nesneleri süslü parantezlerle yazılır `{}`.

Nesne özellikleri, virgülle ayrılmış olarak ad:değer çiftleri olarak yazılır.

Örnek

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Kendin dene "

Yukarıdaki örnekteki nesnenin (kişinin) 4 özelliği vardır: FirstName, lastName, age ve eyeColor.

Bu eğitimin ilerleyen kısımlarında **nesneler** hakkında daha fazla bilgi edineceksiniz .

Operatör türü

`typeof` Bir JavaScript değişkeninin türünü bulmak için JavaScript operatörünü kullanabilirsiniz .

Operatör `typeof` bir değişkenin veya ifadenin türünü döndürür:

Örnek

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string"
typeof "John Doe"     // Returns "string"
```

Kendin dene "

Örnek

```
typeof 0              // Returns "number"
typeof 314            // Returns "number"
typeof 3.14           // Returns "number"
typeof (3)            // Returns "number"
typeof (3 + 4)        // Returns "number"
```

Kendin dene "

Bu eğitimin ilerleyen kısımlarında **typeof** hakkında daha fazla bilgi edineceksiniz .

Tanımsız

JavaScript'te değeri olmayan bir değişkenin değeri vardır `undefined`. Türü de öyle `undefined`.

Örnek

```
let car;    // Value is undefined, type is undefined
```

Kendin dene "

Değeri olarak ayarlayarak herhangi bir değişken boşaltılabilir `undefined`. Türü de olacaktır `undefined`.

Örnek

```
car = undefined;    // Value is undefined, type is undefined
```

Kendin dene "

Boş Değerler

Boş bir değerin ile hiçbir ilgisi yoktur `undefined`.

Boş bir dizenin hem yasal değeri hem de türü vardır.

Örnek

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

Kendin dene "

Egzersizlerle Kendinizi Test Edin

Egzersiz yapmak:

Aşağıdaki değişkenlerin doğru veri türünü açıklamak için yorumları kullanın:

```
uzunluk = 16 olsun; //
let lastName = "Johnson"; //
sabit x = {
  ad: "John",
  Soyadı: "Doe"
}; //
```

Cevabı gönder "

Egzersizi Başlat

JavaScript İşlevleri

❮ Öncesi Sonraki ❯

JavaScript işlevi, belirli bir görevi gerçekleştirmek için tasarlanmış bir kod bloğudur.

Bir JavaScript işlevi "bir şey" onu çağırdığında (onu çağırdığında) yürütülür.

Örnek

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Kendin dene "

JavaScript İşlev Sözdizimi

`function` Bir JavaScript işlevi anahtar kelimeyle, ardından bir **adla** ve ardından parantezlerle **()** tanımlanır .

İşlev adları harf, rakam, alt çizgi ve dolar işareti içerebilir (değişkenlerle aynı kurallar).

Parantez içinde virgülle ayrılmış parametre adları bulunabilir:
(*parametre1, parametre2, ...*)

İşlev tarafından yürütülecek kod küme parantezleri içine alınır: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Fonksiyon **parametreleri,** fonksiyon tanımında parantezlerin () içinde listelenir.

Fonksiyon **argümanları, fonksiyon** çağrıldığında fonksiyon tarafından alınan değerlerdir .

Fonksiyonun içinde argümanlar (parametreler) yerel değişkenler gibi davranır.

İşlev Çağırma

Fonksiyonun içindeki kod, "bir şey" fonksiyonu çağırdığında (çağırdığında) çalışacaktır :

- Bir olay meydana geldiğinde (kullanıcı bir düğmeye tıkladığında)
- JavaScript kodundan çağrıldığında (çağrıldığında)
- Otomatik olarak (kendi kendine çağrılır)

Bu eğitimin ilerleyen kısımlarında işlev çağırma hakkında daha fazla bilgi edineceksiniz.

İşlev Geri Dönüşü

JavaScript bir ifadeye ulaştığında `return` işlevin yürütülmesi durdurulur.

İşlev bir ifadeden çağrıldıysa, JavaScript, çağrılan ifadeden sonra kodu yürütmek için "geri dönecektir".

İşlevler genellikle bir **dönüş değeri** hesaplar . Dönüş değeri "arayana" geri "döndürülür":

Örnek

İki sayının çarpımını hesaplayın ve sonucu döndürün:

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
// Function returns the product of a and b
  return a * b;
}
```

Kendin dene "

Neden Fonksiyonlar?

İşlevlerle kodu yeniden kullanabilirsiniz

Birçok kez kullanılabilecek kodlar yazabilirsiniz.

Farklı sonuçlar elde etmek için aynı kodu farklı argümanlarla kullanabilirsiniz.

Operatör

() operatörü işlevi çağırır (çağırır):

Örnek

Fahrenheit'i Santigrat'a dönüştürün:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
```

```
let value = toCelsius(77);
```

Kendin dene "

Yanlış parametrelerle bir işleve erişmek yanlış yanıt verebilir:

Örnek

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius();
```

Kendin dene "

() olmadan bir işleve erişmek, işlev sonucunu değil, işlevi döndürür:

Örnek

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius;
```

Kendin dene "

Not

Yukarıdaki örneklerden de gördüğünüz gibi `toCelsius` fonksiyon nesnesini, `toCelsius()` fonksiyonun sonucunu ifade eder.

Değişken Değer Olarak Kullanılan Fonksiyonlar

İşlevler, her tür formülde, atamada ve hesaplamada değişkenleri kullandığınız gibi kullanılabilir.

Örnek

Bir fonksiyonun dönüş değerini saklamak için değişken kullanmak yerine:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

Fonksiyonu doğrudan değişken bir değer olarak kullanabilirsiniz:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Kendin dene "

Bu eğitimin ilerleyen kısımlarında işlevler hakkında daha fazla bilgi edineceksiniz.

Yerel Değişkenler

Bir JavaScript işlevi içinde bildirilen değişkenler, işlev için **YEREL hale gelir.**

Yerel değişkenlere yalnızca fonksiyonun içinden erişilebilir.

Örnek

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Kendin dene "

Yerel değişkenler yalnızca fonksiyonların içinde tanındığından aynı isimdeki değişkenler farklı fonksiyonlarda kullanılabilir.

Yerel değişkenler, bir işlev başlatıldığında oluşturulur ve işlev tamamlandığında silinir.

Egzersizlerle Kendinizi Test Edin

Egzersiz yapmak:

adlı işlevi yürütün `myFunction`.

```
işlev işlevim() {
  alarm("Merhaba Dünya!");
}
                ;
```

Cevabı gönder "

Egzersizi Başlat

JavaScript Nesneleri

❮ Öncesi Sonraki ❯

Gerçek Hayattaki Nesneler, Özellikler ve Yöntemler

Gerçek hayatta araba bir **nesnedir** .

Bir arabanın ağırlık ve renk gibi **özellikleri** ve çalıştırma ve durdurma gibi **yöntemleri vardır:**

Nesne	Özellikler	Yöntemler
	araba.adı = Fiat araba.model = 500 araba.ağırlık = 850kg araba.renk = beyaz	car.start() car.drive() car.brake() car.stop()

Tüm arabalar aynı **özelliklere** sahiptir ancak özellik **değerleri** arabadan arabaya farklılık gösterir.

Tüm arabalarda aynı **yöntemler vardır ancak yöntemler farklı zamanlarda** uygulanır .

JavaScript Nesneleri

JavaScript değişkenlerinin veri değerleri için kapsayıcılar olduğunu zaten öğrendiniz.

Bu kod, araba adlı bir **değişkene basit bir değer** (Fiat) atar :

```
let car = "Fiat";
```

Kendin dene "

Nesneler de değişkendir. Ancak nesneler birçok değer içerebilir.

Bu kod, car adlı bir **değişkene birçok değer** (Fiat, 500, beyaz) atar:

```
const car = {type:"Fiat", model:"500", color:"white"};
```

Kendin dene "

Değerler ad:değer çiftleri (iki nokta üst üste işaretiyle ayrılmış ad ve değer) olarak yazılır .

`Nesneleri const` anahtar sözcüğüyle bildirmek yaygın bir uygulamadır .

`Const'u` nesnelerle kullanma hakkında daha fazla bilgiyi şu bölümden öğrenebilirsiniz: JS Const .

Nesne Tanımı

Bir nesne değişmezi ile bir JavaScript nesnesi tanımlarsınız (ve yaratırsınız):

Örnek

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Kendin dene "

Boşluklar ve satır sonları önemli değildir. Bir nesne tanımı birden fazla satıra yayılabilir:

Örnek

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Kendin dene "

Nesne Özellikleri

JavaScript nesnelerindeki ad **:değer çiftlerine özellikler** denir :

Mülk	Mülk değeri
ilk adı	John
soy isim	Doe
yaş	50
göz rengi	mavi

Nesne Özelliklerine Erişim

Nesne özelliklerine iki şekilde erişebilirsiniz:

```
objectName.propertyName
```

veya

```
objectName["propertyName"]
```

Örnek 1

```
person.lastName;
```

Kendin dene "

Örnek2

```
person["lastName"];
```

Kendin dene "

JavaScript nesneleri, özellikler adı verilen adlandırılmış değerlerin kapsayıcılarıdır .

Nesne Yöntemleri

Nesnelerin de yöntemleri olabilir .

Yöntemler nesneler üzerinde gerçekleştirilebilen eylemlerdir .

Yöntemler özelliklerde **işlev tanımları** olarak saklanır .

Mülk	Mülk değeri
ilk adı	John
soy isim	Doe
yaş	50
göz rengi	mavi
Ad Soyad	function() {return this.firstName + " " + this.lastName;}

Yöntem, özellik olarak saklanan bir işlevdir.

Örnek

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

Yukarıdaki örnekte, **kişi nesnesini** `this` ifade eder :

this.firstName, kişiliğin ilkAd özelliği anlamına gelir .

this.lastName, kişinin lastName özelliği anlamına gelir .

Bu nedir ?

JavaScript'te `this` anahtar kelime bir **nesneyi** ifade eder .

Hangi `this` nesnenin nasıl çağrıldığına (kullanıldığına veya çağrıldığına) bağlıdır .

Anahtar kelime `this` , nasıl kullanıldığına bağlı olarak farklı nesneleri ifade eder:

Bir nesne yönteminde, `this` nesneyi ifade eder .
Alone, **global nesneyi** `this` ifade eder .
Bir fonksiyonda, **global nesneyi** `this` ifade eder .
Bir fonksiyonda, katı modda `this` , `undefined` .
Bir etkinlikte, etkinliği alan **öğeyi** `this` ifade eder .
`call()` , `apply()` , ve gibi yöntemler **herhangi bir** nesneye `bind()` başvurabilir . `this`

Not

`this` bir değişken değildir. Bu bir anahtar kelimedir. değerini değiştiremezsiniz `this` .

Ayrıca bakınız:

Bu Eğitimde JavaScript

Bu Anahtar **Kelime**

Bir fonksiyon tanımında `this` fonksiyonun "sahibi"ni ifade eder.

Yukarıdaki örnekte, işlevin "sahibi" olan **kişi** `this` nesnesidir . `fullName`

Başka bir deyişle **bu nesnenin** özelliği `this.firstName` anlamına gelir . `firstName`

Bu Eğitimde JavaScript `this` hakkında daha fazla bilgi edinin .

Nesne Yöntemlerine Erişim

Bir nesne yöntemine aşağıdaki sözdizimiyle erişirsiniz:

```
objectName.methodName()
```

Örnek

```
name = person.fullName();
```

Kendin dene "

() parantezleri olmadan bir yönteme erişirseniz , **işlev tanımını** döndürür :

Örnek

```
name = person.fullName;
```

Kendin dene "

Dizeleri, Sayıları ve Booleanları Nesne Olarak Bildirmeyin!

Bir JavaScript değişkeni " " anahtar sözcüğüyle bildirildiğinde `new` değişken bir nesne olarak oluşturulur:

```
x = new String();        // Declares x as a String object
y = new Number();        // Declares y as a Number object
z = new Boolean();       // Declares z as a Boolean object
```

`String` , `Number` ve nesnelerinden kaçının `Boolean` . Kodunuzu karmaşıklaştırır ve yürütme hızını yavaşlatırlar.

Bu eğitimin ilerleyen kısımlarında nesneler hakkında daha fazla bilgi edineceksiniz.

Egzersizlerle Kendinizi Test Edin

Egzersiz yapmak:

`"John"` Nesneden bilgi çıkararak uyarı verin `person` .

```
sabit kişi = {
  ad: "John",
  Soyadı: "Doe"
};

uyarı(                    );
```

Cevabı gönder "

Egzersizi Başlat

JavaScript Events

‹ Previous Next ›

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an `onclick` attribute (with code), is added to a `<button>` element:

Example

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

Try it Yourself »

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using `this.innerHTML`):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

Try it Yourself »

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

```
<button onclick="displayDate()">The time is?</button>
```

Try it Yourself »

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

The list is much longer: W3Schools JavaScript Reference HTML DOM Events.

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

You will learn a lot more about events and event handlers in the HTML DOM chapters.

JavaScript Dizeleri

❮ Öncesi Sonraki ❯

Dizeler metni depolamak içindir

Dizeler tırnak işaretleri ile yazılır

Alıntıları Kullanma

Bir JavaScript dizesi, tırnak içine yazılan sıfır veya daha fazla karakterden oluşur.

Örnek

```
let text = "John Doe";
```

Kendin dene "

Tek veya çift tırnak kullanabilirsiniz:

Örnek

```
let carName1 = "Volvo XC60";  // Double quotes
let carName2 = 'Volvo XC60';  // Single quotes
```

Kendin dene "

Not

Tek veya çift tırnakla oluşturulan dizeler aynı şekilde çalışır.

İkisi arasında hiçbir fark yoktur.

Alıntılar İçinde Alıntılar

Bir dizenin içindeki tırnak işaretlerini, dizeyi çevreleyen tırnak işaretleri ile eşleşmedikleri sürece kullanabilirsiniz:

Örnek

```
let answer1 = "It's alright";
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';
```

Kendin dene "

Şablon Dizeleri

Şablonlar ES6 (JavaScript 2016) ile tanıtıldı.

Şablonlar, geri tırnak işaretleri içine alınmış dizelerdir ("Bu bir şablon dizesidir").

Şablonlar bir dize içinde tek ve çift tırnaklara izin verir:

Örnek

```
let text = `He's often called "Johnny"`;
```

Kendin dene "

Not

Şablonlar Internet Explorer'da desteklenmez.

IP uzunluğu

Bir dizenin uzunluğunu bulmak için yerleşik `length` özelliği kullanın:

Örnek

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
```

Kendin dene "

Kaçış Karakterleri

Dizelerin tırnak içinde yazılması gerektiğinden, JavaScript bu dizeyi yanlış anlayacaktır:

```
let text = "We are the so-called "Vikings" from the north.";
```

Dize "Biz sözdeyiz" diye kesilecek.

Bu sorunu çözmek için ters eğik çizgi kaçış karakterini kullanabilirsiniz .

Ters eğik çizgi kaçış karakteri (`\`) özel karakterleri dize karakterlerine dönüştürür:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Örnekler

\" bir dizeye çift tırnak ekler:

```
let text = "We are the so-called \"Vikings\" from the north.";
```

Kendin dene "

\' bir dizeye tek bir tırnak işareti ekler:

```
let text= 'It\'s alright.';
```

Kendin dene "

\\ dizeye ters eğik çizgi ekler:

```
let text = "The character \\ is called backslash.";
```

Kendin dene "

JavaScript'te altı kaçış dizisi daha geçerlidir:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

Not

Yukarıdaki 6 kaçış karakteri başlangıçta daktiloları, teletipleri ve faks makinelerini kontrol etmek için tasarlandı. HTML'de hiçbir anlam ifade etmiyorlar.

Uzun Çizgileri Kırmak

Okunabilirlik açısından programcılar genellikle uzun kod satırlarından kaçınmak isterler.

Bir ifadeyi bölmenin güvenli bir yolu bir operatörün peşindedir:

Örnek

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Kendin dene "

Bir dizeyi bölmenin güvenli bir yolu dize eklemeyi kullanmaktır:

Örnek

```
document.getElementById("demo").innerHTML = "Hello " +
"Dolly!";
```

Kendin dene "

Şablon Dizeleri

Şablonlar ES6 (JavaScript 2016) ile tanıtıldı.

Şablonlar, geri tırnak işaretleri içine alınmış dizelerdir ("Bu bir şablon dizesidir").

Şablonlar çok satırlı dizelere izin verir:

Örnek

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Kendin dene "

Not

Şablonlar Internet Explorer'da desteklenmez.

Nesne Olarak JavaScript Dizeleri

Normalde, JavaScript dizeleri değişmez değerlerden oluşturulan ilkel değerlerdir:

```
let x = "John";
```

Ancak dizeler aynı zamanda şu anahtar kelimeye sahip nesneler olarak da tanımlanabilir `new`:

```
let y = new String("John");
```

Örnek

```
let x = "John";
let y = new String("John");
```

Kendin dene "

Strings nesneleri oluşturmayın.

Anahtar kelime `new` kodu karmaşıklaştırır ve yürütme hızını yavaşlatır.

String nesneleri beklenmeyen sonuçlar üretebilir:

Operatörü kullanırken `==` x ve y **eşittir** :

```
let x = "John";
let y = new String("John");
```

Kendin dene "

Operatörü kullanırken `===` x ve y **eşit değildir** :

```
let x = "John";
let y = new String("John");
```

Kendin dene "

`(x==y)` ve arasındaki farka dikkat edin `(x===y)` .

`(x == y)` doğru ya da yanlış?

```
let x = new String("John");
let y = new String("John");
```

Kendin dene "

`(x === y)` doğru ya da yanlış?

```
let x = new String("John");
let y = new String("John");
```

Kendin dene "

İki JavaScript nesnesinin karşılaştırılması **her zaman false** değerini döndürür .

Tam Dize Referansı

Tam bir String referansı için şu adrese gidin:

JavaScript Dizesi Referansını Tamamlayın .

Referans, tüm dize özelliklerinin ve yöntemlerinin açıklamalarını ve örneklerini içerir.

JavaScript Number Methods

‹ Previous Next ›

JavaScript Number Methods

These **number methods** can be used on all JavaScript numbers:

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a number written in exponential notation
toFixed()	Returns a number written with a number of decimals
toPrecision()	Returns a number written with a specified length
valueOf()	Returns a number as a number

The toString() Method

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

```
let x = 123;
x.toString();
(123).toString();
(100 + 23).toString();
```

Try it Yourself »

The toExponential() Method

`toExponential()` returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

Example

```
let x = 9.656;
x.toExponential(2);
x.toExponential(4);
x.toExponential(6);
```

Try it Yourself »

The parameter is optional. If you don't specify it, JavaScript will not round the number.

The toFixed() Method

`toFixed()` returns a string, with the number written with a specified number of decimals:

Example

```
let x = 9.656;
x.toFixed(0);
x.toFixed(2);
x.toFixed(4);
x.toFixed(6);
```

Try it Yourself »

`toFixed(2)` is perfect for working with money.

The toPrecision() Method

`toPrecision()` returns a string, with a number written with a specified length:

Example

```
let x = 9.656;
x.toPrecision();
x.toPrecision(2);
x.toPrecision(4);
x.toPrecision(6);
```

Try it Yourself »

The valueOf() Method

`valueOf()` returns a number as a number.

Example

```
let x = 123;
x.valueOf();
(123).valueOf();
(100 + 23).valueOf();
```

Try it Yourself »

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The `valueOf()` method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

All JavaScript data types have a `valueOf()` and a `toString()` method.

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert a variable to a number:

Method	Description
Number()	Returns a number converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns a whole number

The methods above are not **number** methods. They are **global** JavaScript methods.

The Number() Method

The `Number()` method can be used to convert JavaScript variables to numbers:

Example

```
Number(true);
Number(false);
Number("10");
Number("  10");
Number("10  ");
Number(" 10  ");
Number("10.33");
Number("10,33");
Number("10 33");
Number("John");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

The Number() Method Used on Dates

`Number()` can also convert a date to a number.

Example

```
Number(new Date("1970-01-01"))
```

Try it Yourself »

Note

The `Date()` method returns the number of milliseconds since 1.1.1970.

The number of milliseconds between 1970-01-02 and 1970-01-01 is 86400000:

Example

```
Number(new Date("1970-01-02"))
```

Try it Yourself »

Example

```
Number(new Date("2017-09-30"))
```

Try it Yourself »

The parseInt() Method

`parseInt()` parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

```
parseInt("-10");
parseInt("-10.33");
parseInt("10");
parseInt("10.33");
parseInt("10 20 30");
parseInt("10 years");
parseInt("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

The parseFloat() Method

`parseFloat()` parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

```
parseFloat("10");
parseFloat("10.33");
parseFloat("10 20 30");
parseFloat("10 years");
parseFloat("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

Number Object Methods

These **object methods** belong to the **Number** object:

Method	Description
Number.isInteger()	Returns true if the argument is an integer
Number.isSafeInteger()	Returns true if the argument is a safe integer
Number.parseFloat()	Converts a string to a number
Number.parseInt()	Converts a string to a whole number

Number Methods Cannot be Used on Variables

The number methods above belong to the JavaScript **Number Object**.

These methods can only be accessed like `Number.isInteger()`.

Using X.isInteger() where X is a variable, will result in an error:

`TypeError X.isInteger is not a function`.

The Number.isInteger() Method

The `Number.isInteger()` method returns `true` if the argument is an integer.

Example

```
Number.isInteger(10);
Number.isInteger(10.5);
```

Try it Yourself »

The Number.isSafeInteger() Method

A safe integer is an integer that can be exactly represented as a double precision number.

The `Number.isSafeInteger()` method returns `true` if the argument is a safe integer.

Example

```
Number.isSafeInteger(10);
Number.isSafeInteger(12345678901234567890);
```

Try it Yourself »

Safe integers are all integers from $-(2^{53} - 1)$ to $+(2^{53} - 1)$.
This is safe: 9007199254740991. This is not safe: 9007199254740992.

The Number.parseFloat() Method

`Number.parseFloat()` parses a string and returns a number.

Spaces are allowed. Only the first number is returned:

Example

```
Number.parseFloat("10");
Number.parseFloat("10.33");
Number.parseFloat("10 20 30");
Number.parseFloat("10 years");
Number.parseFloat("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

Note

The **Number** methods `Number.parseInt()` and `Number.parseFloat()`

are the same as the

Global methods `parseInt()` and `parseFloat()`.

The purpose is modularization of globals (to make it easier to use the same JavaScript code outside the browser).

The Number.parseInt() Method

`Number.parseInt()` parses a string and returns a whole number.

Spaces are allowed. Only the first number is returned:

Example

```
Number.parseInt("-10");
Number.parseInt("-10.33");
Number.parseInt("10");
Number.parseInt("10.33");
Number.parseInt("10 20 30");
Number.parseInt("10 years");
Number.parseInt("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

Complete JavaScript Number Reference

For a complete Number reference, visit our:

Complete JavaScript Number Reference.

The reference contains descriptions and examples of all Number properties and methods.

JavaScript Number Properties

❮ Previous Next ❯

Property	Description
EPSILON	The difference between 1 and the smallest number > 1.
MAX_VALUE	The largest number possible in JavaScript
MIN_VALUE	The smallest number possible in JavaScript
MAX_SAFE_INTEGER	The maximum safe integer (2^{53} - 1)
MIN_SAFE_INTEGER	The minimum safe integer -(2^{53} - 1)
POSITIVE_INFINITY	Infinity (returned on overflow)
NEGATIVE_INFINITY	Negative infinity (returned on overflow)
NaN	A "Not-a-Number" value

JavaScript EPSILON

`Number.EPSILON` is the difference between the smallest floating point number greater than 1 and 1.

Example

```
let x = Number.EPSILON;
```

Try it Yourself »

Note

`Number.EPSILON` is an ES6 feature.

It does not work in Internet Explorer.

JavaScript MAX_VALUE

`Number.MAX_VALUE` is a constant representing the largest possible number in JavaScript.

Example

```
let x = Number.MAX_VALUE;
```

Try it Yourself »

Number Properties Cannot be Used on Variables

Number properties belong to the JavaScript **Number Object**.

These properties can only be accessed as `Number.MAX_VALUE`.

Using x.MAX_VALUE, where x is a variable or a value, will return `undefined`:

Example

```
let x = 6;
x.MAX_VALUE
```

Try it Yourself »

JavaScript MIN_VALUE

`Number.MIN_VALUE` is a constant representing the lowest possible number in JavaScript.

Example

```
let x = Number.MIN_VALUE;
```

Try it Yourself »

JavaScript MAX_SAFE_INTEGER

`Number.MAX_SAFE_INTEGER` represents the maximum safe integer in JavaScript.

`Number.MAX_SAFE_INTEGER` is $(2^{53} - 1)$.

Example

```
let x = Number.MAX_SAFE_INTEGER;
```

Try it Yourself »

JavaScript MIN_SAFE_INTEGER

`Number.MIN_SAFE_INTEGER` represents the minimum safe integer in JavaScript.

`Number.MIN_SAFE_INTEGER` is $-(2^{53} - 1)$.

Example

```
let x = Number.MIN_SAFE_INTEGER;
```

Try it Yourself »

Note

`MAX_SAFE_INTEGER` and `MIN_SAFE_INTEGER` are ES6 features.

They do not work in Internet Explorer.

JavaScript POSITIVE_INFINITY

Example

```
let x = Number.POSITIVE_INFINITY;
```

Try it Yourself »

`POSITIVE_INFINITY` is returned on overflow:

```
let x = 1 / 0;
```

Try it Yourself »

JavaScript NEGATIVE_INFINITY

Example

```
let x = Number.NEGATIVE_INFINITY;
```

Try it Yourself »

`NEGATIVE_INFINITY` is returned on overflow:

```
let x = -1 / 0;
```

Try it Yourself »

JavaScript NaN - Not a Number

`NaN` is a JavaScript reserved word for a number that is not a legal number.

Examples

```
let x = Number.NaN;
```

Try it Yourself »

Trying to do arithmetic with a non-numeric string will result in `NaN` (Not a Number):

```
let x = 100 / "Apple";
```

Try it Yourself »

Complete JavaScript Number Reference

For a complete Number reference, visit our:

Complete JavaScript Number Reference.

The reference contains descriptions and examples of all Number properties and methods.

JavaScript Arrays

‹ Previous Next ›

An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

It is a common practice to declare arrays with the `const` keyword.

Learn more about `const` with arrays in the chapter: JS Array Const.

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

Try it Yourself »

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Try it Yourself »

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
const cars = new Array("Saab", "Volvo", "BMW");
```

Try it Yourself »

The two examples above do exactly the same.

There is no need to use `new Array()`.

For simplicity, readability and execution speed, use the array literal method.

Accessing Array Elements

You access an array element by referring to the **index number**:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Try it Yourself »

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

This statement changes the value of the first element in `cars`:

```
cars[0] = "Opel";
```

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Try it Yourself »

Converting an Array to a String

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

Banana,Orange,Apple,Mango

Try it Yourself »

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Try it Yourself »

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, `person[0]` returns John:

Array:

```
const person = ["John", "Doe", 46];
```

Try it Yourself »

Objects use **names** to access its "members". In this example, `person.firstName` returns John:

Object:

```
const person = {firstName:"John", lastName:"Doe", age:46};
```

Try it Yourself »

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

Array methods are covered in the next chapters.

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

Try it Yourself »

The `length` property is always one more than the highest array index.

Accessing the First Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

Try it Yourself »

Accessing the Last Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Try it Yourself »

Looping Array Elements

One way to loop through an array, is using a `for` loop:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

Try it Yourself »

You can also use the `Array.forEach()` function:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

let text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

Try it Yourself »

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits.push("Lemon");  // Adds a new element (Lemon) to fruits
```

Try it Yourself »

New element can also be added to an array using the `length` property:

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[fruits.length] = "Lemon";  // Adds "Lemon" to fruits
```

Try it Yourself »

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon";  // Creates undefined "holes" in fruits
```

Try it Yourself »

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

Example

```
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
person.length;    // Will return 3
person[0];        // Will return "John"
```

Try it Yourself »

WARNING !!
If you use named indexes, JavaScript will redefine the array to an object.

After that, some array methods and properties will produce **incorrect results**.

Example:

```
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
```

```
person["age"] = 46;
person.length;     // Will return 0
person[0];         // Will return undefined
```

Try it Yourself »

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

JavaScript new Array()

JavaScript has a built-in array constructor `new Array()`.

But you can safely use `[]` instead.

These two different statements both create a new empty array named points:

```
const points = new Array();
const points = [];
```

These two different statements both create a new array containing 6 numbers:

```
const points = new Array(40, 100, 1, 5, 25, 10);
const points = [40, 100, 1, 5, 25, 10];
```

Try it Yourself »

The `new` keyword can produce some unexpected results:

```
// Create an array with three elements:
const points = new Array(40, 100, 1);
```

Try it Yourself »

```
// Create an array with two elements:
const points = new Array(40, 100);
```

Try it Yourself »

```
// Create an array with one element ???
const points = new Array(40);
```

Try it Yourself »

A Common Error

```
const points = [40];
```

is not the same as:

```
const points = new Array(40);
```

```
// Create an array with one element:
const points = [40];
```

Try it Yourself »

```
// Create an array with 40 undefined elements:
const points = new Array(40);
```

Try it Yourself »

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator `typeof` returns "`object`":

```
const fruits = ["Banana", "Orange", "Apple"];
let type = typeof fruits;
```

Try it Yourself »

The typeof operator returns object because a JavaScript array is an object.

Solution 1:

To solve this problem ECMAScript 5 (JavaScript 2009) defined a new method `Array.isArray()`:

```
Array.isArray(fruits);
```

Try it Yourself »

Solution 2:

The `instanceof` operator returns true if an object is created by a given constructor:

```
const fruits = ["Banana", "Orange", "Apple"];

fruits instanceof Array;
```

Try it Yourself »

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

Test Yourself With Exercises

Exercise:

Get the value "`Volvo`" from the `cars` array.

```
const cars = ["Saab", "Volvo", "BMW"];
let x =        ;
```

Submit Answer »

Start the Exercise

JavaScript Array Methods

❮ Previous Next ❯

Basic Array Methods

Array length
Array toString()
Array at()
Array join()
Array pop()
Array push()
Array shift()
Array unshift()
Array delete()
Array concat()
Array copyWithin()
Array flat()
Array splice()
Array toSpliced()
Array slice()

See Also:

Search Methods
Sort Methods
Iteration Methods

JavaScript Array length

The `length` property returns the length (size) of an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let size = fruits.length;
```

Try it Yourself »

JavaScript Array toString()

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

```
Banana,Orange,Apple,Mango
```

Try it Yourself »

JavaScript Array at()

ES2022 intoduced the array method `at()`:

Examples

Get the third element of fruits using at():

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.at(2);
```

Try it Yourself »

Get the third element of fruits using []:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[2];
```

Try it Yourself »

The `at()` method returns an indexed element from an array.

The `at()` method returns the same as `[]`.

The `at()` method is supported in all modern browsers since March 2022:

				
Chrome 92	Edge 92	Firefox 90	Safari 15.4	Opera 78
Apr 2021	Jul 2021	Jul 2021	Mar 2022	Aug 2021

Note

Many languages allows `negative bracket indexing` like [-1] to access elements from the end of an object / array / string.

This is not possible in JavaScript, because [] is used for accessing both arrays and objects. obj[-1] refers to the value of key -1, not to the last property of the object.

The `at()` method was introduced in ES2022 to solve this problem.

JavaScript Array join()

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result:

Banana * Orange * Apple * Mango

Try it Yourself »

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

JavaScript Array pop()

The `pop()` method removes the last element from an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

Try it Yourself »

The `pop()` method returns the value that was "popped out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

Try it Yourself »

JavaScript Array push()

The `push()` method adds a new element to an array (at the end):

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

Try it Yourself »

The `push()` method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

Try it Yourself »

Shifting Elements

Shifting is equivalent to popping, but working on the first element instead of the last.

JavaScript Array shift()

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

Try it Yourself »

The `shift()` method returns the value that was "shifted out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```

Try it Yourself »

JavaScript Array unshift()

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Try it Yourself »

The `unshift()` method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Try it Yourself »

Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element
[1] is the second
[2] is the third ...

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

Try it Yourself »

JavaScript Array length

The `length` property provides an easy way to append a new element to an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";
```

Try it Yourself »

JavaScript Array delete()

Warning !

Using `delete()` leaves `undefined` holes in the array.

Use pop() or shift() instead.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];
```

Try it Yourself »

Merging Arrays (Concatenating)

In programming languages, concatenation means joining strings end-to-end.

Concatenation "snow" and "ball" gives "snowball".

Concatenating arrays means joining arrays end-to-end.

JavaScript Array concat()

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);
```

Try it Yourself »

Note

The `concat()` method does not change the existing arrays. It always returns a new array.

The `concat()` method can take any number of array arguments.

Example (Merging Three Arrays)

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3);
```

Try it Yourself »

The `concat()` method can also take strings as arguments:

Example (Merging an Array with Values)

```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");
```

Try it Yourself »

Array copyWithin()

The `copyWithin()` method copies array elements to another position in an array:

Examples

Copy to index 2, all elements from index 0:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.copyWithin(2, 0);
```

Try it Yourself »

Copy to index 2, the elements from index 0 to 2:

```
const fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi"];
fruits.copyWithin(2, 0, 2);
```

Try it Yourself »

Note

The `copyWithin()` method overwrites the existing values.

The `copyWithin()` method does not add items to the array.

The `copyWithin()` method does not change the length of the array.

Flattening an Array

Flattening an array is the process of reducing the dimensionality of an array.

Flattening is useful when you want to convert a multi-dimensional array into a one-dimensional array.

JavaScript Array flat()

ES2019 Introduced the Array flat() method.

The `flat()` method creates a new array with sub-array elements concatenated to a specified depth.

Example

```
const myArr = [[1,2],[3,4],[5,6]];
const newArr = myArr.flat();
```

Try it Yourself »

Browser Support

JavaScript Array `flat()` is supported in all modern browsers since January 2020:

				
Chrome 69	Edge 79	Firefox 62	Safari 12	Opera 56
Sep 2018	Jan 2020	Sep 2018	Sep 2018	Sep 2018

Splicing and Slicing Arrays

The `splice()` method adds new items to an array.

The `slice()` method slices out a piece of an array.

JavaScript Array splice()

The `splice()` method can be used to add new items to an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Try it Yourself »

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```

Try it Yourself »

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);
```

Try it Yourself »

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

JavaScript Array toSpliced()

ES2023 added the Array toSpliced() method as a safe way to splice an array without altering the original array.

The difference between the new **toSpliced()** method and the old **splice()** method is that the new method creates a new array, keeping the original array unchanged, while the old method altered the original array.

Example

```
const months = ["Jan", "Feb", "Mar", "Apr"];
const spliced = months.toSpliced(0, 1);
```

Try it Yourself »

JavaScript Array slice()

The `slice()` method slices out a piece of an array into a new array:

Example

Slice out a part of an array starting from array element 1 ("Orange"):

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
```

Try it Yourself »

Note

The `slice()` method creates a new array.

The `slice()` method does not remove any elements from the source array.

Example

Slice out a part of an array starting from array element 3 ("Apple"):

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(3);
```

Try it Yourself »

The `slice()` method can take two arguments like `slice(1, 3)`.

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
```

Try it Yourself »

If the end argument is omitted, like in the first examples, the `slice()` method slices out the rest of the array.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);
```

Try it Yourself »

Automatic toString()

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Try it Yourself »

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
```

Try it Yourself »

Note

All JavaScript objects have a toString() method.

Searching Arrays

Searching arrays are covered in the next chapter of this tutorial.

Sorting Arrays

Sorting arrays covers the methods used to sort arraysg.

Iterating Arrays

Iterating arrays covers methods that operate on all array elements.

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

Test Yourself With Exercises

Exercise:

Use the correct Array method to remove the **last item** of the `fruits` array.

```
const fruits = ["Banana", "Orange", "Apple"];
                ;
```

Submit Answer »

Start the Exercise

JavaScript Array Search

❮ Previous Next ❯

Array Find and Search Methods

Array indexOf()
Array lastIndexOf()
Array includes()

Array find()
Array findIndex()
Array findLast()
Array findLastIndex()

See Also:

Basic Methods
Sort Methods
Iteration Methods

JavaScript Array indexOf()

The `indexOf()` method searches an array for an element value and returns its position.

Note: The first item has position 0, the second item has position 1, and so on.

Example

Search an array for the item "Apple":

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.indexOf("Apple") + 1;
```

Try it Yourself »

Syntax

```
array.indexOf(item, start)
```

item	Required. The item to search for.
start	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

`Array.indexOf()` returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

JavaScript Array lastIndexOf()

`Array.lastIndexOf()` is the same as `Array.indexOf()`, but returns the position of the last occurrence of the specified element.

Example

Search an array for the item "Apple":

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.lastIndexOf("Apple") + 1;
```

Try it Yourself »

Syntax

```
array.lastIndexOf(item, start)
```

item	Required. The item to search for
start	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning

JavaScript Array includes()

ECMAScript 2016 introduced `Array.includes()` to arrays. This allows us to check if an element is present in an array (including NaN, unlike indexOf).

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.includes("Mango"); // is true
```

Try it Yourself »

Syntax

```
array.includes(search-item)
```

Array.includes() allows to check for NaN values. Unlike Array.indexOf().

Browser Support

`includes()` is an ECMAScript 2016 feature.

ES 2016 is fully supported in all modern browsers since March 2017:

Chrome 52	Edge 15	Firefox 52	Safari 10.1	Opera 39
Jul 2016	Apr 2017	Mar 2017	May 2017	Aug 2016

`includes()` is not supported in Internet Explorer.

JavaScript Array find()

The `find()` method returns the value of the first array element that passes a test function.

This example finds (returns the value of) the first element that is larger than 18:

Example

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.find(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

Browser Support

`find()` is an ES6 feature (JavaScript 2015).

ES6 is fully supported in all modern browsers since June 2017:

Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

`find()` is not supported in Internet Explorer.

JavaScript Array findIndex()

The `findIndex()` method returns the index of the first array element that passes a test function.

This example finds the index of the first element that is larger than 18:

Example

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
```

```
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

Browser Support

`findIndex()` is an ES6 feature (JavaScript 2015).

ES6 is fully supported in all modern browsers since June 2017:

				
Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

`findIndex()` is not supported in Internet Explorer.

JavaScript Array findLast() Method

ES2023 added the findLast() method that will start from the end of an array and return the value of the first element that satisfies a condition.

Example

```
const temp = [27, 28, 30, 40, 42, 35, 30];
let high = temp.findLast(x => x > 40);
```

Try it Yourself »

Browser Support

`findLast()` is an ES2023 feature.

It is supported in all modern browsers since July 2023:

				
Chrome 110	Edge 110	Firefox 115	Safari 16.4	Opera 96
Feb 2023	Feb 2023	Jul 2023	Mar 2023	May 2023

JavaScript Array findLastIndex() Method

The findLastIndex() method finds the index of the last element that satisfies a condition.

Example

```
const temp = [27, 28, 30, 40, 42, 35, 30];
let pos = temp.findLastIndex(x => x > 40);
```

Try it Yourself »

Browser Support

`findLastIndex()` is an ES2023 feature.

It is supported in all modern browsers since July 2023:

				
Chrome 110	Edge 110	Firefox 115	Safari 16.4	Opera 96
Feb 2023	Feb 2023	Jul 2023	Mar 2023	May 2023

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

JavaScript Sorting Arrays

❮ Previous Next ❯

Alpabetic Sort

Array sort()
Array reverse()
Array toSorted()
Array toReversed()
Sorting Objects

Numeric Sort

Numeric Sort
Random Sort
Math.min()
Math.max()
Home made Min()
Home made Max()

See Also:

Basic Methods
Search Methods
Iteration Methods

Sorting an Array

The `sort()` method sorts an array alphabetically:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

Try it Yourself »

Reversing an Array

The `reverse()` method reverses the elements in an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.reverse();
```

Try it Yourself »

By combining `sort()` and `reverse()`, you can sort an array in descending order:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
fruits.reverse();
```

Try it Yourself »

JavaScript Array toSorted() Method

ES2023 added the `toSorted()` method as a safe way to sort an array without altering the original array.

The difference between `toSorted()` and `sort()` is that the first method creates a new array, keeping the original array unchanged, while the last method alters the original array.

Example

```
const months = ["Jan", "Feb", "Mar", "Apr"];
const sorted = months.toSorted();
```

Try it Yourself »

JavaScript Array toReversed() Method

ES2023 added the `toReversed()` method as a safe way to reverse an array without altering the original array.

The difference between `toReversed()` and `reverse()` is that the first method creates a new array, keeping the original array unchanged, while the last method alters the original array.

Example

```
const months = ["Jan", "Feb", "Mar", "Apr"];
const reversed = months.toReversed();
```

Try it Yourself »

Numeric Sort

By default, the `sort()` function sorts values as **strings**.

This works well for strings ("Apple" comes before "Banana").

If numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the `sort()` method will produce incorrect result when sorting numbers.

You can fix this by providing a **compare function**:

Example

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
```

Try it Yourself »

Use the same trick to sort an array descending:

Example

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});
```

Try it Yourself »

The Compare Function

The purpose of the compare function is to define an alternative sort order.

The compare function should return a negative, zero, or positive value, depending on the arguments:

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

When the `sort()` function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

If the result is negative, `a` is sorted before `b`.

If the result is positive, `b` is sorted before `a`.

If the result is 0, no changes are done with the sort order of the two values.

Example:

The compare function compares all the values in the array, two values at a time `(a, b)`.

When comparing 40 and 100, the `sort()` method calls the compare function(40, 100).

The function calculates 40 - 100 `(a - b)`, and since the result is negative (-60), the sort function will sort 40 as a value lower than 100.

You can use this code snippet to experiment with numerically and alphabetically sorting:

```
<button onclick="myFunction1()">Sort Alphabetically</button>
<button onclick="myFunction2()">Sort Numerically</button>

<p id="demo"></p>

<script>
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points;

function myFunction1() {
  points.sort();
  document.getElementById("demo").innerHTML = points;
}

function myFunction2() {
  points.sort(function(a, b){return a - b});
  document.getElementById("demo").innerHTML = points;
}
</script>
```

Try it Yourself »

Sorting an Array in Random Order

Using a sort function, like explained above, you can sort an numeric array in random order

Example

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(){return 0.5 - Math.random()});
```

Try it Yourself »

The Fisher Yates Method

The points.sort() method in the example above is not accurate. It will favor some numbers over others.

The most popular correct method, is called the Fisher Yates shuffle, and was introduced in data science as early as 1938!

In JavaScript the method can be translated to this:

Example

```
const points = [40, 100, 1, 5, 25, 10];

for (let i = points.length -1; i > 0; i--) {
  let j = Math.floor(Math.random() * (i+1));
  let k = points[i];
  points[i] = points[j];
  points[j] = k;
}
```

Try it Yourself »

Find the Lowest (or Highest) Array Value

There are no built-in functions for finding the max or min value in an array.

To find the lowest or highest value you have 3 options:

- Sort the array and read the first or last element
- Use Math.min() or Math.max()
- Write a home made function

Find Min or Max with sort()

After you have sorted an array, you can use the index to obtain the highest and lowest values.

Sort Ascending:

Example

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
// now points[0] contains the lowest value
// and points[points.length-1] contains the highest value
```

Try it Yourself »

Sort Descending:

Example

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});
// now points[0] contains the highest value
// and points[points.length-1] contains the lowest value
```

Try it Yourself »

Note

Sorting a whole array is a very inefficient method if you only want to find the highest (or lowest) value.

Using Math.min() on an Array

You can use `Math.min.apply` to find the lowest number in an array:

Example

```
function myArrayMin(arr) {
  return Math.min.apply(null, arr);
}
```

Try it Yourself »

`Math.min.apply(null, [1, 2, 3])` is equivalent to `Math.min(1, 2, 3)`.

Using Math.max() on an Array

You can use `Math.max.apply` to find the highest number in an array:

Example

```
function myArrayMax(arr) {
  return Math.max.apply(null, arr);
}
```

Try it Yourself »

`Math.max.apply(null, [1, 2, 3])` is equivalent to `Math.max(1, 2, 3)`.

JavaScript Array Minimum Method

There is no built-in function for finding the lowest value in a JavaScript array.

The fastest code to find the lowest number is to use a **home made** method.

This function loops through an array comparing each value with the lowest value found:

Example (Find Min)

```
function myArrayMin(arr) {
  let len = arr.length;
  let min = Infinity;
  while (len--) {
    if (arr[len] < min) {
      min = arr[len];
    }
  }
  return min;
}
```

Try it Yourself »

JavaScript Array Maximum Method

There is no built-in function for finding the highest value in a JavaScript array.

The fastest code to find the highest number is to use a **home made** method.

This function loops through an array comparing each value with the highest value found:

Example (Find Max)

```
function myArrayMax(arr) {
  let len = arr.length;
  let max = -Infinity;
  while (len--) {
    if (arr[len] > max) {
      max = arr[len];
    }
  }
  return max;
}
```

Try it Yourself »

Sorting Object Arrays

JavaScript arrays often contain objects:

Example

```
const cars = [
  {type:"Volvo", year:2016},
  {type:"Saab", year:2001},
  {type:"BMW", year:2010}
];
```

Even if objects have properties of different data types, the `sort()` method can be used to sort the array.

The solution is to write a compare function to compare the property values:

Example

```
cars.sort(function(a, b){return a.year - b.year});
```

Try it Yourself »

Comparing string properties is a little more complex:

Example

```
cars.sort(function(a, b){
  let x = a.type.toLowerCase();
  let y = b.type.toLowerCase();
  if (x < y) {return -1;}
  if (x > y) {return 1;}
  return 0;
});
```

Try it Yourself »

Stable Array sort()

ES2019 **revised** the Array `sort()` method.

Before 2019, the specification allowed unstable sorting algorithms such as QuickSort.

After ES2019, browsers must use a stable sorting algorithm:

When sorting elements on a value, the elements must keep their relative position to other elements with the same value.

Example

```
const myArr = [
  {name:"X00",price:100 },
  {name:"X01",price:100 },
  {name:"X02",price:100 },
  {name:"X03",price:100 },
  {name:"X04",price:110 },
  {name:"X05",price:110 },
  {name:"X06",price:110 },
  {name:"X07",price:110 }
];
```

Try it Yourself »

In the example above, when sorting on price, the result is not allowed to come out with the names in an other relative position like this:

```
X01 100
X03 100
X00 100
X03 100
X05 110
X04 110
X06 110
X07 110
```

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

Test Yourself With Exercises

Exercise:

Use the correct Array method to sort the `fruits` array alphabetically.

```
const fruits = ["Banana", "Orange", "Apple", "Kiwi"];
                ;
```

Submit Answer »

Start the Exercise

JavaScript Array Iteration

❮ Previous Next ❯

Array Iteration Methods

Array iteration methods operate on every array item:

Array forEach
Array map()
Array flatMap()
Array filter()
Array reduce()
Array reduceRight()
Array every()
Array some()
Array from()
Array keys()
Array entries()
Array with()
Array Spread (...)

See Also:

Basic Array Methods
Array Search Methods
Array Sort Methods

JavaScript Array forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

Example

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value + "<br>";
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. The example can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value) {
  txt += value + "<br>";
}
```

Try it Yourself »

JavaScript Array map()

The `map()` method creates a new array by performing a function on each array element.

The `map()` method does not execute the function for array elements without values.

The `map()` method does not change the original array.

This example multiplies each array value by 2:

Example

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses only the value parameter, the index and array parameters can be omitted:

Example

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);

function myFunction(value) {
  return value * 2;
}
```

Try it Yourself »

JavaScript Array flatMap()

ES2019 added the Array `flatMap()` method to JavaScript.

The `flatMap()` method first maps all elements of an array and then creates a new array by flattening the array.

Example

```
const myArr = [1, 2, 3, 4, 5, 6];
const newArr = myArr.flatMap((x) => x * 2);
```

Try it Yourself »

Browser Support

JavaScript Array `flatMap()` is supported in all modern browsers since January 2020:

				
Chrome 69	Edge 79	Firefox 62	Safari 12	Opera 56

Sep 2018	Jan 2020	Sep 2018	Sep 2018	Sep 2018

JavaScript Array filter()

The `filter()` method creates a new array with array elements that pass a test.

This example creates a new array from elements with a value larger than 18:

Example

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

In the example above, the callback function does not use the index and array parameters, so they can be omitted:

Example

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value) {
  return value > 18;
}
```

Try it Yourself »

JavaScript Array reduce()

The `reduce()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduce()` method works from left-to-right in the array. See also `reduceRight()`.

The `reduce()` method does not reduce the original array.

This example finds the sum of all numbers in an array:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Try it Yourself »

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

The example above does not use the index and array parameters. It can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);

function myFunction(total, value) {
  return total + value;
}
```

Try it Yourself »

The `reduce()` method can accept an initial value:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction, 100);

function myFunction(total, value) {
  return total + value;
}
```

Try it Yourself »

JavaScript Array reduceRight()

The `reduceRight()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduceRight()` works from right-to-left in the array. See also `reduce()`.

The `reduceRight()` method does not reduce the original array.

This example finds the sum of all numbers in an array:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduceRight(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Try it Yourself »

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

The example above does not use the index and array parameters. It can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduceRight(myFunction);

function myFunction(total, value) {
  return total + value;
}
```

Try it Yourself »

JavaScript Array every()

The `every()` method checks if all array values pass a test.

This example checks if all array values are larger than 18:

Example

```
const numbers = [45, 4, 9, 16, 25];
let allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses the first parameter only (value), the other parameters can be omitted:

Example

```
const numbers = [45, 4, 9, 16, 25];
let allOver18 = numbers.every(myFunction);

function myFunction(value) {
  return value > 18;
}
```

Try it Yourself »

JavaScript Array some()

The `some()` method checks if some array values pass a test.

This example checks if some array values are larger than 18:

Example

```
const numbers = [45, 4, 9, 16, 25];
let someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

JavaScript Array.from()

The `Array.from()` method returns an Array object from any object with a length property or any iterable object.

Example

Create an Array from a String:

```
Array.from("ABCDEFG");
```

Try it Yourself »

Browser Support

`from()` is an ES6 feature (JavaScript 2015).

ES6 is fully supported in all modern browsers since June 2017:

				
Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

`from()` is not supported in Internet Explorer.

JavaScript Array keys()

The `Array.keys()` method returns an Array Iterator object with the keys of an array.

Example

Create an Array Iterator object, containing the keys of the array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = fruits.keys();

for (let x of keys) {
  text += x + "<br>";
}
```

Try it Yourself »

Browser Support

`keys()` is an ES6 feature (JavaScript 2015).

ES6 is fully supported in all modern browsers since June 2017:

				
Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

`keys()` is not supported in Internet Explorer.

JavaScript Array entries()

Example

Create an Array Iterator, and then iterate over the key/value pairs:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.entries();

for (let x of f) {
  document.getElementById("demo").innerHTML += x;
}
```

Try it Yourself »

The `entries()` method returns an Array Iterator object with key/value pairs:

[0, "Banana"]
[1, "Orange"]
[2, "Apple"]
[3, "Mango"]

The `entries()` method does not change the original array.

Browser Support

`entries()` is an ES6 feature (JavaScript 2015).

ES6 is fully supported in all modern browsers since June 2017:

Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

`entries()` is not supported in Internet Explorer.

JavaScript Array with() Method

ES2023 added the Array with() method as a safe way to update elements in an array without altering the original array.

Example

```
const months = ["Januar", "Februar", "Mar", "April"];
const myMonths = months.with(2, "March");
```

Try it Yourself »

JavaScript Array Spread (...)

The ... operator expands an iterable (like an array) into more elements:

Example

```
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const q3 = ["Jul", "Aug", "Sep"];
const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];
```

Try it Yourself »

Browser Support

`...` is an ES6 feature (JavaScript 2015).

ES6 is fully supported in all modern browsers since June 2017:

Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

`...` is not supported in Internet Explorer.

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

JavaScript Array Const

❮ Previous Next ❯

ECMAScript 2015 (ES6)

In 2015, JavaScript introduced an important new keyword: `const`.

It has become a common practice to declare arrays using `const`:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Cannot be Reassigned

An array declared with `const` cannot be reassigned:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

Try it Yourself »

Arrays are Not Constants

The keyword `const` is a little misleading.

It does NOT define a constant array. It defines a constant reference to an array.

Because of this, we can still change the elements of a constant array.

Elements Can be Reassigned

You can change the elements of a constant array:

Example

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

Try it Yourself »

Browser Support

The `const` keyword is not supported in Internet Explorer 10 or earlier.

The following table defines the first browser versions with full support for the `const` keyword:

Chrome 49	IE 11 / Edge	Firefox 36	Safari 10	Opera 36
Mar, 2016	Oct, 2013	Feb, 2015	Sep, 2016	Mar, 2016

Assigned when Declared

JavaScript `const` variables must be assigned a value when they are declared:

Meaning: An array declared with `const` must be initialized when it is declared.

Using `const` without initializing the array is a syntax error:

Example

This will not work:

```
const cars;
cars = ["Saab", "Volvo", "BMW"];
```

Arrays declared with `var` can be initialized at any time.

You can even use the array before it is declared:

Example

This is OK:

```
cars = ["Saab", "Volvo", "BMW"];
var cars;
```

Try it Yourself »

Const Block Scope

An array declared with `const` has **Block Scope**.

An array declared in a block is not the same as an array declared outside the block:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
// Here cars[0] is "Saab"
{
  const cars = ["Toyota", "Volvo", "BMW"];
  // Here cars[0] is "Toyota"
}
// Here cars[0] is "Saab"
```

Try it Yourself »

An array declared with `var` does not have block scope:

Example

```
var cars = ["Saab", "Volvo", "BMW"];
// Here cars[0] is "Saab"
{
  var cars = ["Toyota", "Volvo", "BMW"];
  // Here cars[0] is "Toyota"
}
// Here cars[0] is "Toyota"
```

Try it Yourself »

You can learn more about Block Scope in the chapter: JavaScript Scope.

Redeclaring Arrays

Redeclaring an array declared with `var` is allowed anywhere in a program:

Example

```
var cars = ["Volvo", "BMW"];   // Allowed
var cars = ["Toyota", "BMW"];  // Allowed
cars = ["Volvo", "Saab"];      // Allowed
```

Redeclaring or reassigning an array to `const`, in the same scope, or in the same block, is not allowed:

Example

```
var cars = ["Volvo", "BMW"];     // Allowed
const cars = ["Volvo", "BMW"];   // Not allowed
{
  var cars = ["Volvo", "BMW"];   // Allowed
  const cars = ["Volvo", "BMW"]; // Not allowed
}
```

Redeclaring or reassigning an existing `const` array, in the same scope, or in the same block, is not allowed:

Example

```
const cars = ["Volvo", "BMW"];   // Allowed
const cars = ["Volvo", "BMW"];   // Not allowed
var cars = ["Volvo", "BMW"];     // Not allowed
cars = ["Volvo", "BMW"];         // Not allowed

{
  const cars = ["Volvo", "BMW"]; // Allowed
  const cars = ["Volvo", "BMW"]; // Not allowed
  var cars = ["Volvo", "BMW"];   // Not allowed
  cars = ["Volvo", "BMW"];       // Not allowed
}
```

Redeclaring an array with `const`, in another scope, or in another block, is allowed:

Example

```
const cars = ["Volvo", "BMW"];   // Allowed
{
  const cars = ["Volvo", "BMW"]; // Allowed
}
{
```

```
  const cars = ["Volvo", "BMW"]; // Allowed
}
```

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

❮ Previous Next ❯

JavaScript Classes

‹ Previous Next ›

ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

JavaScript Classes are templates for JavaScript Objects.

JavaScript Class Syntax

Use the keyword `class` to create a class.

Always add a method named `constructor()`:

Syntax

```
class ClassName {
  constructor() { ... }
}
```

Example

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

The example above creates a class named "Car".

The class has two initial properties: "name" and "year".

A JavaScript class is **not** an object.

It is a **template** for JavaScript objects.

Using a Class

When you have a class, you can use the class to create objects:

Example

```
const myCar1 = new Car("Ford", 2014);
const myCar2 = new Car("Audi", 2019);
```

Try it Yourself »

The example above uses the **Car class** to create two **Car objects**.

The constructor method is called automatically when a new object is created.

The Constructor Method

The constructor method is a special method:

- It has to have the exact name "constructor"
- It is executed automatically when a new object is created
- It is used to initialize object properties

If you do not define a constructor method, JavaScript will add an empty constructor method.

Class Methods

Class methods are created with the same syntax as object methods.

Use the keyword `class` to create a class.

Always add a `constructor()` method.

Then add any number of methods.

Syntax

```
class ClassName {
  constructor() { ... }
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
}
```

Create a Class method named "age", that returns the Car age:

Example

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    const date = new Date();
    return date.getFullYear() - this.year;
  }
}

const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
```

Try it Yourself »

You can send parameters to Class methods:

Example

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}

const date = new Date();
let year = date.getFullYear();

const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML=
"My car is " + myCar.age(year) + " years old.";
```

Try it Yourself »

Browser Support

The following table defines the first browser version with full support for Classes in JavaScript:

Chrome 49	Edge 12	Firefox 45	Safari 9	Ope
Mar, 2016	Jul, 2015	Mar, 2016	Oct, 2015	Mar

"use strict"

The syntax in classes must be written in "strict mode".

You will get an error if you do not follow the "strict mode" rules.

Example

In "strict mode" you will get an error if you use a variable without declaring it:

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    // date = new Date();  // This will not work
    const date = new Date(); // This will work
    return date.getFullYear() - this.year;
  }
}
```

Try it Yourself »

Learn more about "strict mode" in: JS Strict Mode.

JavaScript Class Inheritance

‹ Previous Next ›

Class Inheritance

To create a class inheritance, use the `extends` keyword.

A class created with a class inheritance inherits all the methods from another class:

Example

Create a class named "Model" which will inherit the methods from the "Car" class:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

Try it Yourself »

The `super()` method refers to the parent class.

By calling the `super()` method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.

Getters and Setters

Classes also allows you to use getters and setters.

It can be smart to use getters and setters for your properties, especially if you want to do something special with the value before returning them, or before you set them.

To add getters and setters in the class, use the `get` and `set` keywords.

Example

Create a getter and a setter for the "carname" property:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
```

Try it Yourself »

Note: even if the getter is a method, you do not use parentheses when you want to get the property value.

The name of the getter/setter method cannot be the same as the name of the property, in this case `carname`.

Many programmers use an underscore character `_` before the property name to separate the getter/setter from the actual property:

Example

You can use the underscore character to separate the getter/setter from the actual property:

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");
```

```
document.getElementById("demo").innerHTML = myCar.carname;
```

Try it Yourself »

To use a *setter*, use the same syntax as when you set a property value, without parentheses:

Example

Use a setter to change the carname to "Volvo":

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");
myCar.carname = "Volvo";
document.getElementById("demo").innerHTML = myCar.carname;
```

Try it Yourself »

Hoisting

Unlike functions, and other JavaScript declarations, class declarations are not hoisted.

That means that you must declare a class before you can use it:

Example

```
//You cannot use the class yet.
//myCar = new Car("Ford") will raise an error.

class Car {
  constructor(brand) {
    this.carname = brand;
  }
}

//Now you can use the class:
const myCar = new Car("Ford")
```

Try it Yourself »

Note: For other declarations, like functions, you will NOT get an error when you try to use it before it is declared, because the default behavior of JavaScript declarations are hoisting (moving the declaration to the top).

JavaScript Static Methods

❮ Previous Next ❯

Static class methods are defined on the class itself.

You cannot call a `static` method on an object, only on an object class.

Example

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello() {
    return "Hello!!";
  }
}

const myCar = new Car("Ford");

// You can call 'hello()' on the Car Class:
document.getElementById("demo").innerHTML = Car.hello();

// But NOT on a Car Object:
// document.getElementById("demo").innerHTML = myCar.hello();
// this will raise an error.
```

Try it Yourself »

If you want to use the myCar object inside the `static` method, you can send it as a parameter:

Example

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
const myCar = new Car("Ford");
document.getElementById("demo").innerHTML = Car.hello(myCar);
```

Try it Yourself »

JavaScript Callbacks

❮ Previous Next ❯

"I will call back later!"

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

Function Sequence

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

This example will end up displaying "Goodbye":

Example

```
function myFirst() {
  myDisplayer("Hello");
}

function mySecond() {
  myDisplayer("Goodbye");
}

myFirst();
mySecond();
```

Try it Yourself »

This example will end up displaying "Hello":

Example

```
function myFirst() {
  myDisplayer("Hello");
}

function mySecond() {
  myDisplayer("Goodbye");
}

mySecond();
myFirst();
```

Try it Yourself »

Sequence Control

Sometimes you would like to have better control over when to execute a function.

Suppose you want to do a calculation, and then display the result.

You could call a calculator function (`myCalculator`), save the result, and then call another function (`myDisplayer`) to display the result:

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = myCalculator(5, 5);
myDisplayer(result);
```

Try it Yourself »

Or, you could call a calculator function (`myCalculator`), and let the calculator function call the display function (`myDisplayer`):

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  myDisplayer(sum);
}

myCalculator(5, 5);
```

Try it Yourself »

The problem with the first example above, is that you have to call two functions to display the result.

The problem with the second example, is that you cannot prevent the calculator function from displaying the result.

Now it is time to bring in a callback.

JavaScript Callbacks

A callback is a function passed as an argument to another function.

Using a callback, you could call the calculator function (`myCalculator`) with a callback (`myCallback`), and let the calculator function run the callback after the calculation is finished:

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Try it Yourself »

In the example above, `myDisplayer` is a called a **callback function**.

It is passed to `myCalculator()` as an **argument**.

Note

When you pass a function as an argument, remember not to use parenthesis.

Right: myCalculator(5, 5, myDisplayer);

Wrong: ~~myCalculator(5, 5, myDisplayer());~~

Example

```
// Create an Array
const myNumbers = [4, 1, -20, -7, 5, 9, -6];

// Call removeNeg with a callback
const posNumbers = removeNeg(myNumbers, (x) => x >= 0);

// Display Result
document.getElementById("demo").innerHTML = posNumbers;

// Keep only positive numbers
function removeNeg(numbers, callback) {
  const myArray = [];
  for (const x of numbers) {
    if (callback(x)) {
      myArray.push(x);
    }
  }
  return myArray;
```

```
}
```

Try it Yourself »

In the example above, `(x) => x >= 0` is a **callback function**.

It is passed to `removeNeg()` as an **argument**.

When to Use a Callback?

The examples above are not very exciting.

They are simplified to teach you the callback syntax.

Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

Asynchronous functions are covered in the next chapter.

Asynchronous JavaScript

❮ Previous Next ❯

"I will finish later!"

Functions running in **parallel** with other functions are called **asynchronous**

A good example is JavaScript setTimeout()

Asynchronous JavaScript

The examples used in the previous chapter, was very simplified.

The purpose of the examples was to demonstrate the syntax of callback functions:

Example

```
function myDisplayer(something) {
  document.getElementById("demo").innerHTML = something;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Try it Yourself »

In the example above, `myDisplayer` is the name of a function.

It is passed to `myCalculator()` as an argument.

In the real world, callbacks are most often used with asynchronous functions.

A typical example is JavaScript `setTimeout()`.

Waiting for a Timeout

When using the JavaScript function `setTimeout()`, you can specify a callback function to be executed on time-out:

Example

```
setTimeout(myFunction, 3000);

function myFunction() {
  document.getElementById("demo").innerHTML = "I love You !!";
}
```

Try it Yourself »

In the example above, `myFunction` is used as a callback.

`myFunction` is passed to `setTimeout()` as an argument.

3000 is the number of milliseconds before time-out, so `myFunction()` will be called after 3 seconds.

Note

When you pass a function as an argument, remember not to use parenthesis.

Right: setTimeout(myFunction, 3000);

Wrong: ~~setTimeout(myFunction(), 3000)~~;

Instead of passing the name of a function as an argument to another function, you can always pass a whole function instead:

Example

```
setTimeout(function() { myFunction("I love You !!!"); }, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}
```

Try it Yourself »

In the example above, `function(){ myFunction("I love You !!!"); }` is used as a callback. It is a complete function. The complete function is passed to setTimeout() as an argument.

3000 is the number of milliseconds before time-out, so `myFunction()` will be called after 3 seconds.

Waiting for Intervals:

When using the JavaScript function `setInterval()`, you can specify a callback function to be executed for each interval:

Example

```
setInterval(myFunction, 1000);

function myFunction() {
  let d = new Date();
  document.getElementById("demo").innerHTML=
  d.getHours() + ":" +
  d.getMinutes() + ":" +
  d.getSeconds();
}
```

Try it Yourself »

In the example above, `myFunction` is used as a callback.

`myFunction` is passed to `setInterval()` as an argument.

1000 is the number of milliseconds between intervals, so `myFunction()` will be called every second.

Callback Alternatives

With asynchronous programming, JavaScript programs can start long-running tasks, and continue running other tasks in paralell.

But, asynchronus programmes are difficult to write and difficult to debug.

Because of this, most modern asynchronous JavaScript methods don't use callbacks. Instead, in JavaScript, asynchronous programming is solved using **Promises** instead.

Note

You will learn about promises in the next chapter of this tutorial.

JavaScript Promises

‹ Previous Next ›

"I Promise a Result!"

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is an Object that links Producing code and Consuming code

JavaScript Promise Object

A Promise contains both the producing code and calls to the consuming code:

Promise Syntax

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

When the producing code obtains the result, it should call one of the two callbacks:

When	Call
Success	myResolve(result value)
Error	myReject(error object)

Promise Object Properties

A JavaScript Promise object can be:

- Pending
- Fulfilled
- Rejected

The Promise object supports two properties: **state** and **result**.

While a Promise object is "pending" (working), the result is undefined.

When a Promise object is "fulfilled", the result is a value.

When a Promise object is "rejected", the result is an error object.

myPromise.state	myPromise.result
"pending"	undefined
"fulfilled"	a result value
"rejected"	an error object

You cannot access the Promise properties **state** and **result**.

You must use a Promise method to handle promises.

Promise How To

Here is how to use a Promise:

```
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Promise.then() takes two arguments, a callback for success and another for failure.

Both are optional, so you can add a callback for success or failure only.

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;

// The producing code (this may take some time)

  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

Try it Yourself »

JavaScript Promise Examples

To demonstrate the use of promises, we will use the callback examples from the previous chapter:

- Waiting for a Timeout
- Waiting for a File

Waiting for a Timeout

Example Using Callback

```
setTimeout(function() { myFunction("I love You !!!"); }, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}
```

Try it Yourself »

Example Using Promise

```
let myPromise = new Promise(function(myResolve, myReject) {
  setTimeout(function() { myResolve("I love You !!"); }, 3000);
});

myPromise.then(function(value) {
  document.getElementById("demo").innerHTML = value;
});
```

Try it Yourself »

Waiting for a file

Example using Callback

```
function getFile(myCallback) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.html");
  req.onload = function() {
    if (req.status == 200) {
      myCallback(req.responseText);
    } else {
      myCallback("Error: " + req.status);
    }
  }
  req.send();
}

getFile(myDisplayer);
```

Try it Yourself »

Example using Promise

```
let myPromise = new Promise(function(myResolve, myReject) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.htm");
  req.onload = function() {
    if (req.status == 200) {
      myResolve(req.response);
```

```
    } else {
      myReject("File not Found");
    }
  };
  req.send();
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

Try it Yourself »

Browser Support

ECMAScript 2015, also known as ES6, introduced the JavaScript Promise object.

The following table defines the first browser version with full support for Promise objects:

				
Chrome 33	Edge 12	Firefox 29	Safari 7.1	Opera 20
Feb, 2014	Jul, 2015	Apr, 2014	Sep, 2014	Mar, 2014

JavaScript Async

‹ Previous Next ›

"async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

Async Syntax

The keyword `async` before a function makes the function return a promise:

Example

```
async function myFunction() {
  return "Hello";
}
```

Is the same as:

```
function myFunction() {
  return Promise.resolve("Hello");
}
```

Here is how to use the Promise:

```
myFunction().then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Example

```
async function myFunction() {
  return "Hello";
}
myFunction().then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

Try it Yourself »

Or simpler, since you expect a normal value (a normal response, not an error):

Example

```
async function myFunction() {
  return "Hello";
}
myFunction().then(
  function(value) {myDisplayer(value);}
);
```

Try it Yourself »

Await Syntax

The `await` keyword can only be used inside an `async` function.

The `await` keyword makes the function pause the execution and wait for a resolved promise before it continues:

```
let value = await promise;
```


Example

Let's go slowly and learn how to use it.

Basic Syntax

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve, reject) {
    resolve("I love You !!");
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

myDisplay();
```

Try it Yourself »

The two arguments (resolve and reject) are pre-defined by JavaScript.

We will not create them, but call one of them when the executor function is ready.

Very often we will not need a reject function.

Example without reject

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    resolve("I love You !!");
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

myDisplay();
```

Try it Yourself »

Waiting for a Timeout

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    setTimeout(function() {resolve("I love You !!");}, 3000);
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

myDisplay();
```

Try it Yourself »

Waiting for a File

```
async function getFile() {
  let myPromise = new Promise(function(resolve) {
    let req = new XMLHttpRequest();
    req.open('GET', "mycar.html");
    req.onload = function() {
      if (req.status == 200) {
        resolve(req.response);
      } else {
        resolve("File not Found");
      }
    };
    req.send();
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

getFile();
```

Try it Yourself »

Browser Support

ECMAScript 2017 introduced the JavaScript keywords `async` and `await`.

The following table defines the first browser version with full support for both:

				
Chrome 55	Edge 15	Firefox 52	Safari 11	Ope
Dec, 2016	Apr, 2017	Mar, 2017	Sep, 2017	Dec