## Javascript nedir ?

JavaScript, tarayıcıda çalışan bir dildir. İçeriği günceller, form verilerini doğrular, multimedyayı kontrol eder ve statik HTML web sayfalarını etkileşimli web sayfalarına dönüştürür. Günümüzde JavaScript yalnızca tarayıcılarda değil, sunucuda veya JavaScript Motoru olan herhangi bir cihazda da çalışabilir .

Örneğin, Node.js , sunucuda yürütülen JavaScript tabanlı bir frameworktür. Bir tarayıcı ve IDE gereklidir.

ECMAScript; JavaScript öncelikle tarayıcılarda yürütülmek üzere geliştirildi. Dünyada farklı birçok tarayıcı var. Bu nedenle, tüm tarayıcılarda aynı işlevselliği elde etmek için JavaScript kodunun yürütülmesini standartlaştırmaya ihtiyaç vardı.

Orijinal JavaScript ES1 ES2 ES3 (1997-1999)

İlk Ana Revizyon ES5 (2009)

İkinci Revizyon ES6 (2015)

Yıllık Eklemeler (2016, 2017, 2018)

1. Web sayfalarının içeriğini tanımlamak için HTML

2. Web sayfalarının düzenini belirtmek için CSS

3. Web sayfalarının davranışını programlamak için JavaScript

## JavaScript Variables

ES6, iki önemli yeni JavaScript anahtar sözcüğünü tanıttı: let ve const. Bu iki anahtar kelime ,

JavaScript'te Blok Kapsamı sağlar . Bir { } bloğu içinde bildirilen değişkenlere, bloğun dışından

erişilemez.

**1- Var = Bir değişken bildirir (ES6 öncesi)**

Var Anahtar kelime ile bildirilen değişkenler blok kapsamına sahip olamaz. Bir { } bloğu içinde tanımlanan değişkenlere bloğun dışından erişilebilir. Değişkeni declare etmeden önce kullanabiliriz.

**2- Let = Bir blok değişkeni bildirir (ES6 sonrası)**

let ile tanımlanan değişkenler yeniden declare edilemez.

let ile tanımlanan değişkenler kullanılmadan önce bildirilmelidir.

Let ile tanımlanan değişkenler Block Scope'a sahip olur.

**3- Const = Bir blok sabiti bildirir (ES6 sonrası)**

const ile tanımlanan değişkenler yeniden declare edilemez.

const ile tanımlanan değişkenler yeniden atanamaz.

const ile tanımlanan değişkenlerin Blok Kapsamı vardır.

Bir bildirim anahtar sözcüğü (var, let veya const) olmadan oluşturulan değişkenler, bir işlev içinde oluşturulmuş olsalar bile her zaman globaldir.

Variable Lifetime

Global değişkenler, başka bir sayfaya gittiğinizde veya pencereyi kapattığınızda olduğu gibi, sayfa atılana kadar yaşar.

Yerel değişkenlerin ömrü kısadır. Func. çağrıldığında oluşturulurlar ve func. bittiğinde silinirler.

{var x = 2;}  
// x CAN be used here

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

let x = "John Doe";  
let x = 0;  
// SyntaxError: 'x' has already been declared

var x = "John Doe";  
var x = 0;

var x = 10;  
// Here x is 10  
{var x = 2; // Here x is 2}  
// Here x is 2

let x = 10;  
// Here x is 10  
{let x = 2; // Here x is 2}  
// Here x is 10

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

With **let**, you cannot use a variable before it is declared.

**ReferenceError**: Cannot access 'carName' before initialization

// hem let hem de const derledikten sonra değer atamasına izin vermiyor.

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

const PI;  
PI = 3.14159265359;

**Because of this you can NOT:**

* Reassign a constant value
* Reassign a constant array
* Reassign a constant object

But you CAN:

* Change a constant array // aşağıda sadece array örneği vardır.
* Change a constant object // Objelerde de aynısı geçerlidir

// 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");

const cars = ["Saab", "Volvo", "BMW"];  
cars = ["Toyota", "Volvo", "Audi"];    // ERROR

**Global Değişkenlerden Kaçının**

Global değişkenlerin kullanımını en aza indirin. Bu, tüm veri türlerini, nesneleri ve işlevleri içerir. Global değişkenler ve işlevlerin üzerine diğer komut dosyaları yazılabilir.

// Declare at the beginning  
let firstName, lastName, price, discount, fullPrice;  
// Use later  
firstName = "John";  
lastName = "Doe";  
price = 19.90;  
discount = 0.10;  
fullPrice = price - discount;

// Declare and initiate at the beginning  
let firstName = "",  
let lastName = "",  
let price = 0,  
let discount = 0,  
let fullPrice = 0,  
const myArray = [],  
const myObject = {};

**Don't Use new Object()**

* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use {} instead of new Object()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new Function()

let x1 = "";             // new primitive string  
let x2 = 0;              // new primitive number  
let x3 = false;          // new primitive boolean  
const x4 = {};           // new object  
const x5 = [];           // new array object  
const x6 = /()/;         // new regexp object  
const x7 = function(){}; // new function object

## Primitive Data Types

**String**

**Number;** typeof NaN; // returns "number"

**BigInt**

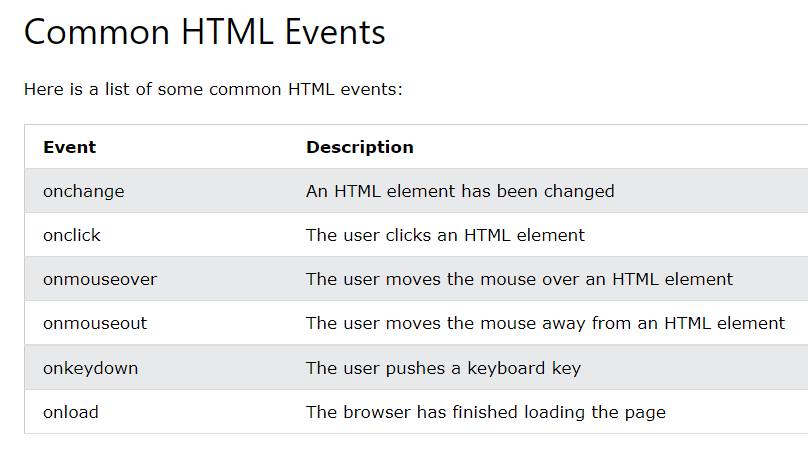
**Boolean**

**Null**; Bir değişkene, o değişkenin şu anda herhangi bir değeri olmadığını, ancak daha sonra sahip olacağını belirtmek için null atanabilir. Null, bir değerin yokluğu anlamına gelir. null nesne türündedir, örneğin typeof null "nesne" döndürür.

**Undefined**; undefined bir simgedir. typeof undefined bir nesne değil, undefined döndürür.

**Symbol**

## JavaScript Events

****

HTML DOM olayları, JavaScript'in bir HTML belgesindeki öğelere farklı olay işleyicileri kaydetmesine izin verir.

Olaylar normalde fonksiyonlarla birlikte kullanılır ve olay meydana gelmeden (örneğin bir kullanıcı bir düğmeyi tıkladığında) fonksiyon yürütülmez.

**...**

## Structural Data Types (reference data types)

Referans veri tipleri bellekteki yere göre karşılaştırma yaparlar. Primitive veri tipleri ise değerlere göre karşılaştırma yaparlar. Bundan dolayı object-array karşılaştırmasında false dönecektir, çünkü bellekteki yerleri farklıdır. Eşitleme durumu ise reference veri tiplerinde aynı bellek yerini göstemesi demektir. Yani bir objeyi başka bir objeye veya değere eşitlersek, o objenin bellekteki yerini referans olarak göstermiş oluyoruz. Primitive veri tiplerinde karşılaştırma ise tuttukları değere göre yapılır. Eşitlemede ise aynı değeri üzerine yazarlar.

**Object,** js ‘de object yoktur, object mantığında çalışan func. lar vardır. (ES6 sonrası)

function Student() {

this.name = 'John';

this.gender = 'M'; }

Student.prototype.age = 15;

var studObj1 = new Student();

alert(studObj1.age); // 15

var studObj2 = new Student();

alert(studObj2.age); // 15

**Date**

**Array**

**JavaScript'te diziler numaralı dizinleri kullanır .**

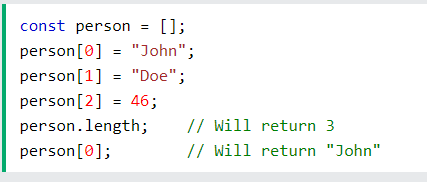
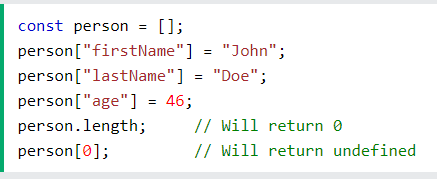
**JavaScript'te nesneler adlandırılmış dizinleri kullanır .**

Diziler, numaralandırılmış dizinleri olan özel bir tür nesnedir.

**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.**

****

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

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

typeof "John"                 // Returns "string"  
typeof 3.14                   // Returns "number"  
typeof NaN                    // Returns "number"  
typeof false                  // Returns "boolean"  
typeof [1,2,3,4]              // Returns "object"  
typeof {name:'John', age:34}  // Returns "object"  
typeof new Date()             // Returns "object"  
typeof function () {}         // Returns "function"  
typeof myCar                  // Returns "undefined" \*  
typeof null                   // Returns "object"

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

## This keyword ?

this belirli bir nesneye işaret eder. Bu nesnenin ne olduğu, 'this' anahtar sözcüğünü içeren bir işlevin nasıl çağrıldığına bağlıdır. This ‘in hangi nesneye atıfta bulunduğunu bilmek için dört kural vardır.

**Global Scope**

**Object's Method**

**Call () veya apply () method**

**Bind() method**

**Global Scope;** 'this' anahtar sözcüğünü içeren bir işlev, **gobal scope** olarakçağrılırsa (fonksiyonun dışında, altında), this o zaman **window object ‘i** işaret eder. **Global Scope**, window object ‘e bağlı anlamına gelir. (this.myVar).

**Object's Method** ‘da ise new keyword u kullanarak bir fonksiyonun nesnesini oluşturursak, o zaman object’s method ’a işaret edecektir. Global olmayacaktır.

**Call () veya apply () method;** call() ve apply()'ın temel amacı, this keyword’ünün **global scope** or as **object's method** olarak mı çağrıldığına bakılmaksızın, o fonksiyona ait olmasını sağlar.

**Bind() method;** bind() yöntemi ECMAScript 5'ten beri tanıtılmıştır. Bir işlev çağrıldığında 'this' bağlamını belirtilen bir nesneye ayarlamak için kullanılabilir.

**Precedence (Öncelik)**

bind()

call() and apply()

Object method

Global scope

**Bind()** kullanarak bir func.’ın callback func. olarak çağrıldığını kontrol edin, Değilse Parametre ile **call()** veya **Apply()** kullanılarak bir func. çağrıldığını kontrol edin, Değilse, bir func.’ın **Object method olarak** çağrıldığını kontrol edin. Değilse, **Global scope** bir fonksiyonun nokta gösterimi olmadan mı yoksa window object’i kullanılarak mı çağrıldığını kontrol edin.

'this' anahtar sözcüğünü içeren bir func, global scope’tan çağrılırsa this, window object’e işaret eder.

<script>

var myVar = 100;

function WhoIsThis() {

var myVar = 200;

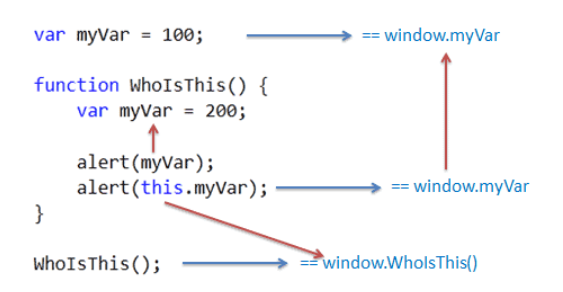
alert(myVar); // 200

alert(this.myVar); // 100

}

WhoIsThis(); // inferred as window.WhoIsThis()

</script>

****

'this', bir iç func.’da kullanılsa bile gloabal window object’ine işaret eder.

var myVar = 100;

function SomeFunction() {

function WhoIsThis() {

var myVar = 200;

alert("myVar = " + myVar); // 200

alert("this.myVar = " + this.myVar); // 100

}

WhoIsThis();

}

SomeFunction();

new anahtar sözcüğünü kullanarak bir func.’ın object’ini oluşturduğunuzda this, o belirli object’i işaret edecektir.

var myVar = 100;

function WhoIsThis() {

this.myVar = 200;

}

var obj1 = new WhoIsThis();

var obj2 = new WhoIsThis();

obj2.myVar = 300;

alert(obj1.myVar); // 200

alert(obj2.myVar); // 300

ilk alert-myVar =300, diğeri ise this olduğu için üst kapsamda ki myVar’a işaret edecektir =200

var myVar = 100;

function WhoIsThis() {

this.myVar = 200;

this.display = function(){

var myVar = 300;

alert("myVar = " + myVar); // 300

alert("this.myVar = " + this.myVar); // 200

};

}

var obj = new WhoIsThis();

obj.display();

=> object literal kullanılarak da bir değişkene atanmış fonksiyonun methodunu çağırırsak; this, local’e işaret edecektir.

var myVar = 100;

function WhoIsThis() {

this.myVar = 200;

}

var obj1 = new WhoIsThis();

var obj2 = new WhoIsThis();

obj2.myVar = 300;

alert(obj1.myVar); // 200

alert(obj2.myVar); // 300

JavaScript'te thisanahtar kelime, nesnenin yöntemiyle birlikte kullanıldığında nesneye atıfta bulunur. thisbir nesneye bağlıdır.

## New keyword ?

**new** **keyword** sözcüğünü kullanarak, tüm veri tiplerine erişim sağlayabiliriz. Bir nevi constructor yapısı oluşturmaya izin verir ve herhangi bir fonksiyonun(variables ‘ın) nesnesini oluşturabiliriz.

Her func.’ın prototype özelliği görünürdür, dışarıdan erişilebilirdir. fakat object’in veya bir class’ın prototype özelliği görünür değildir. New keyword’ü bize bu gizli özelliklere erişmemizi sağlayıp, dışarıdan kullanabilmemizi, ulaşmamızı sağlar.

Mesela static bir prototype’a (func-class-object) e dışarıdan erişim sağlayabiliyoruz. Math methodları gibi. Ancak static olmayan bir method gizli haldedir. Nşa’da erişemeyiz.

**1-** new, empty object oluşturur, ör. nesne = { };

**2-** new, empty object'in görünmez 'prototype' özelliğini, constructor ını görünür ve erişilebilir yapar. (Her **function** görünür 'prototype' özelliğine sahipken, her **object** görünmez 'prototype' özelliğine sahiptir)

**3-** this anahtar sözcüğüyle bildirilen özelliği veya function'ı new object’e bağlar.

function, non-primitive bir değer (özel nesne) döndürürse, **new keyword** o değeri döndürmez.

function MyFunc() {

this.x = 100;

return { a: 123 };

}

var obj1 = new MyFunc();

alert(obj1.x); // undefined

**New keyword**’ü ilkel değer döndüren dönüş ifadesini yok sayar.

function MyFunc() {

this.x = 100;

return 200;

}

var obj = new MyFunc();

alert(obj.x); // 100

**new keyword** sözcüğüyle değişmez sözdizimi(literal syntax) veya yapıcı sözdizimi (constructor syntax) kullanılarak oluşturulan her object, \_\_proto\_\_ objecti oluşturan bir func.’ın prototype objectine işaret eden özelliği içerir.

function Student() {

this.name = 'John';

this.gender = 'M';

}

var studObj = new Student();

console.log(Student.prototype); // object

console.log(studObj.prototype); // undefined

console.log(studObj.\_\_proto\_\_); // object

console.log(typeof Student.prototype); // object

console.log(typeof studObj.\_\_proto\_\_); // object

console.log(Student.prototype === studObj.\_\_proto\_\_ ); // true

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Prototype** özelliği, for..in veya foreach döngüsü kullanılarak yinelenemeyen özel numaralandırılabilir **object** türüdür.

Herhangi bir fonsiyonun NŞA’da durumu böyledir. 2 tane prototype’ı oluşturulmuş.name-age. Hemen altında olan ilk proto ise fonksiyonun methodlarının ve genel JS objesinin bulunduğu yerdir. (varsa constructor’ının bulunduğu yerdir)

Burada showInfos() methodu constructor olarak verilmediği için proto’da yer almıştır.

functionName(className, objectName farketmez hangisiyse) {name: “sddssd”, age: 1655}

**age**:1665

**name**:” sddssd”

**\_\_proto\_\_:**

**showInfos**: f()

**constructor:** f functionName(name,age)

**\_\_proto\_\_:** Object

Burada yok ama bu func.’ın protoytype’ını bir başka bir func’ın prototype’ına atayabiliriz.(sadece değişken değil başka bir func. olabilir veya class)

Mesela boş bir func. açarız. (ya da değişken)

Function Person(){};

Person.prototype = Object.create(functionName.prototype);

**=>**

**//**burada direkt olarak proto çıkmasının sebebi, Person func’ının prototype’nın olmamasından dolayıdır. Boş.

Person(className, objectName farketmez hangisiyse) { }

**//Person’ın prototype’ları olsaydı burada olacaktı.**

**\_\_proto\_\_:** functionName

**age**:1665

**name**:” sddssd”

**\_\_proto\_\_:** Object

**showInfos**: f()

**constructor:** f functionName(name,age)

**\_\_proto\_\_:** Object **//** genel object’e aittir.

Böylelikle 4 katmanlı bir yapı oldu. Sırayla; ilk olarak Person ‘ın kendisi (yani varsa prototype’ları -ki yok. Sonra Person’ın proto’su, functionName’in kendisine bağladığımızı için Object.create ile functionName ‘in kendisi(varsa prototype’ları). Daha sonra functionName’in proto’su ve en sonunda ise genel obje olan JS object.

**1- Person ‘ın kendisi**

**2- Person’ın proto’su = functionName’in kendisi (varsa prototype’ları**

**3- functionName’in proto’su**

**4- Genel JS objesi**

Her object’in prototype’ı, func’ın prototype object’ine bağlıdır. Fonksiyonun prototype’ını değiştirirseniz, değiştirilen prototype’a yalnızca yeni objectler bağlanır. Diğer tüm mevcut objectler hala eski function prototype’ını bağlıdır.

Prototype object JavaScript motoru tarafından iki şeyde kullanılır: 1) bir object’in properties and method larını bulmak 2) JavaScript'te kalıtımı(inheritance) uygulamak.



The built-in primitive types in JavaScript are **functions** only e.g. Object, Boolean, String, Number is built-in JavaScript **functions**. If you write Object in browser's console window and press Enter then you will see the output "**function** Object()".

JavaScript'teki yerleşik ilkel türler yalnızca fonksiyonlardır; Nesne, Boolean, Dize, Sayı yerleşik JavaScript fonksiyonlardır. Tarayıcının konsol penceresine Object yazarsanız ve Enter'a basarsanız, "function Object()" çıktısını göreceksiniz.

## JavaScript Closure

JavaScript değişkenleri yerel veya global kapsama ait olabilir. Global değişkenler kapanışlarla yerel (özel) hale getirilebilir. Closure, bir iç function’ın, dış function geri döndükten sonra bile, dış function’ın değişkenlerine ve parametrelerine her zaman erişimi olduğu anlamına gelir..

Bir fonksiyonun "özel" değişkenlere sahip olmasını mümkün kılar.

function Counter() {

var counter = 0;

function IncreaseCounter() {

return counter += 1;

};

return IncreaseCounter;

}

var counter = Counter();

alert(counter()); // 1

alert(counter()); // 2

Closure, JavaScript'te uygulama ayrıntılarını gizlemek için kullanışlıdır. Başka bir deyişle,

özel değişkenler veya işlevler oluşturmak faydalı olabilir.

var counter = (function() {

var privateCounter = 0;

function changeBy(val) {

privateCounter += val;

}

return {

increment: function() {

changeBy(1);

},

decrement: function() {

changeBy(-1);

},

value: function() {

return privateCounter;

}

};

})();

alert(counter.value()); // 0

counter.increment();

counter.increment();

alert(counter.value()); // 2

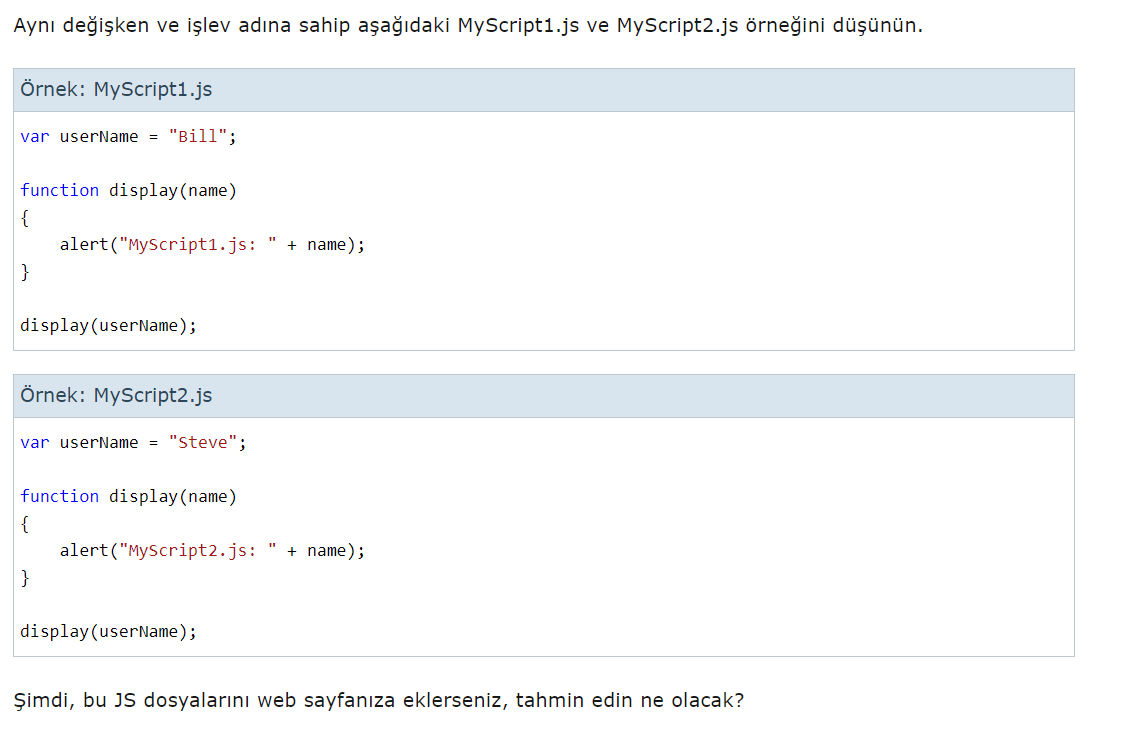
counter.decrement();

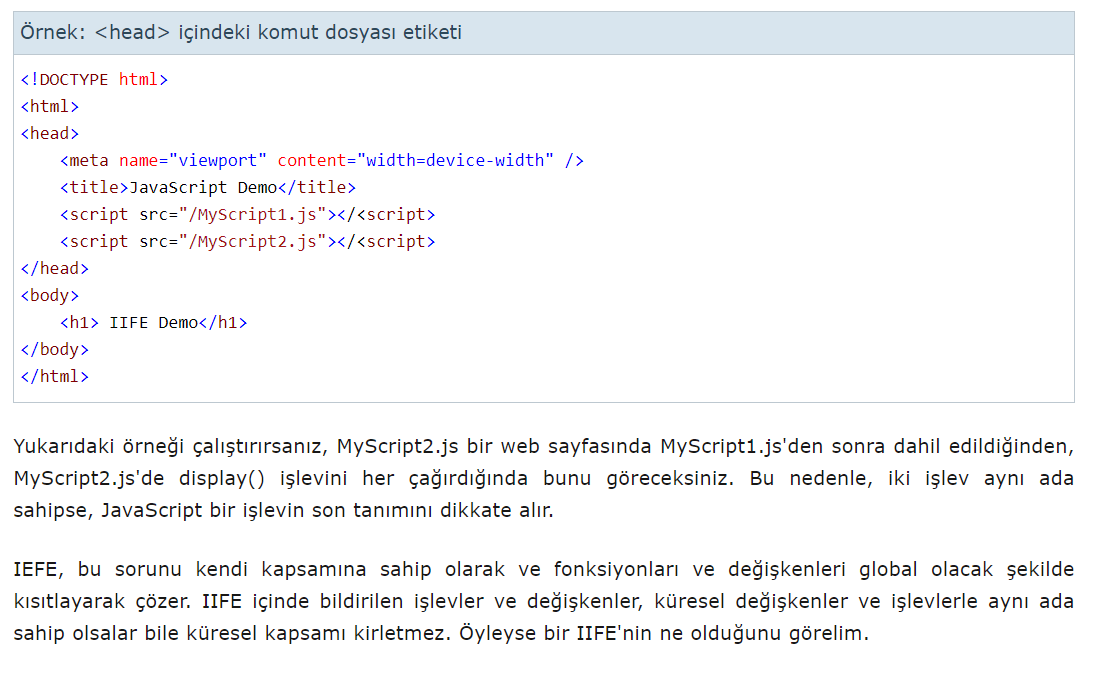
alert(counter.value()); // 1

## Immediately Invoked Function Expression – IIFE

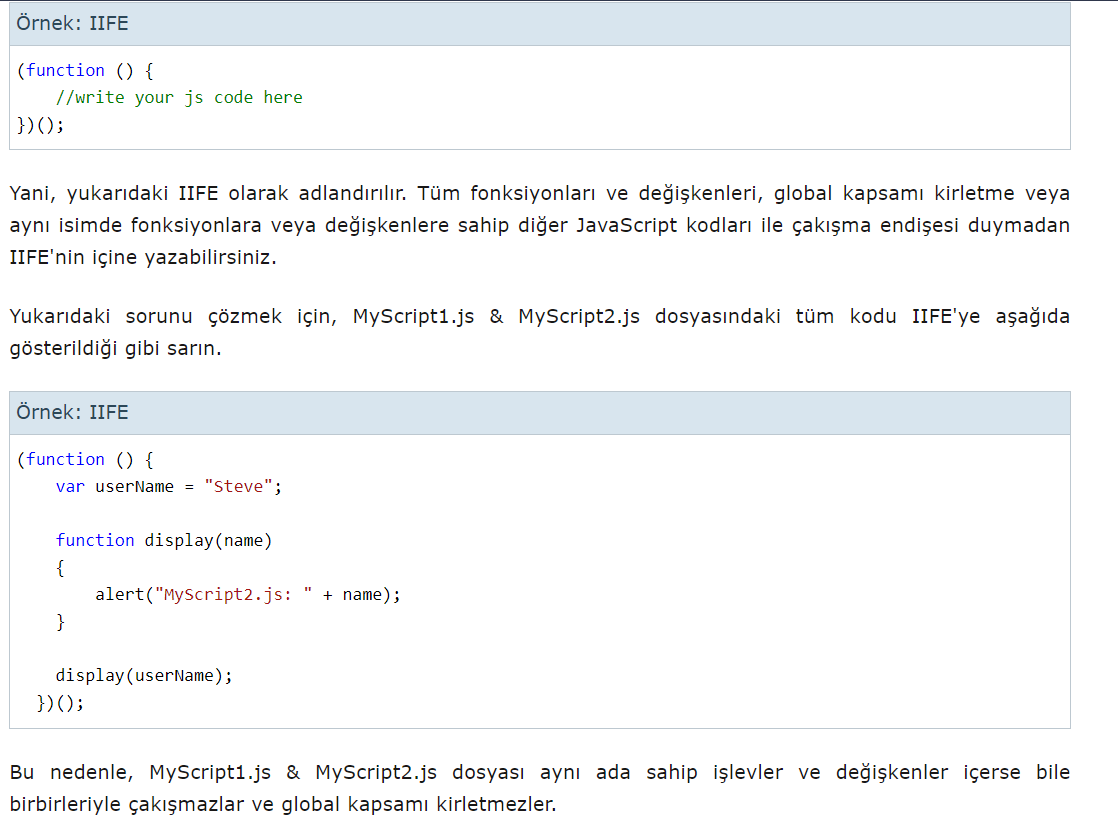
JavaScript'teki bir fonksiyon yerel kapsamı yaratır. Böylece, o fonksiyonun dışından erişilemeyen bir fonksiyonun içinde değişkenler ve fonksiyon tanımlayabilirsiniz. Ancak bazen farkında olmadan değişkenlere ve işlevlere global değişken ve işlev adlarıyla aynı adı vererek genel değişkenleri veya işlevleri yanlışlıkla kirletirsiniz.

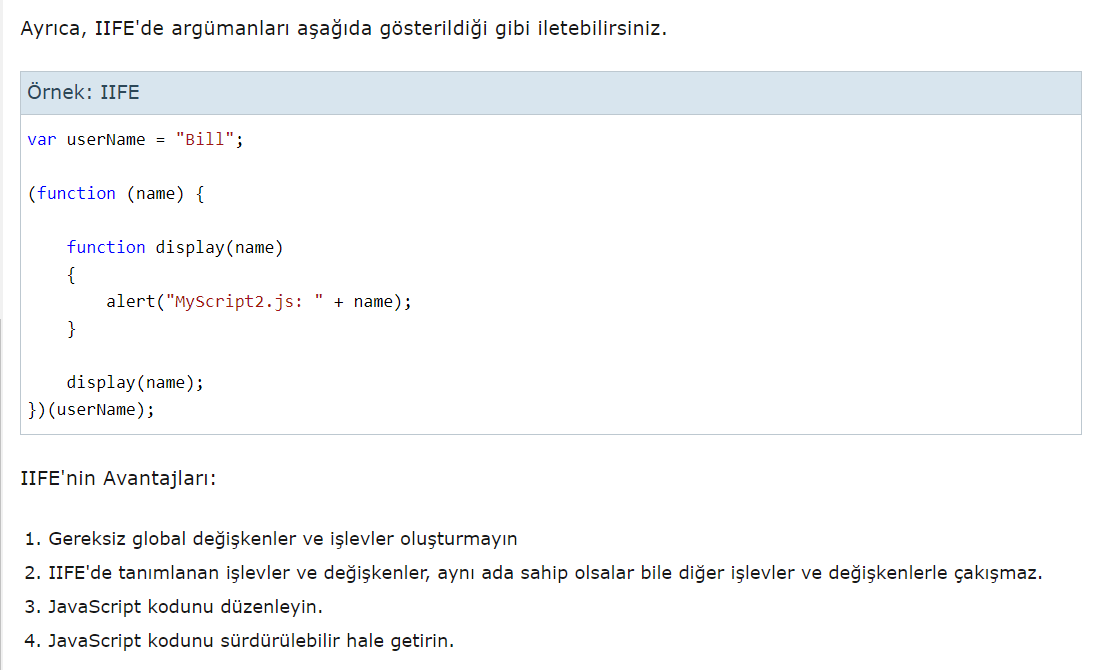
js dosyaları birden fazla geliştirici tarafından yazılmıştır ve bu dosyalar tek bir web sayfasına dahil edilirse, aynı ada sahip iki veya daha fazla işlev veya değişkene sahip olarak global kapsamı kirletecektir.





Adından da anlaşılacağı gibi, IIFE, tanım tamamlandıktan sonra otomatik olarak çağrılan bir işlev ifadesidir . Parantez () IIFE modelinde önemli bir rol oynar. JavaScript'te parantez ifadeler içeremez; yalnızca bir ifade içerebilir.





## JavaScript Scope

Scope determines the accessibility (visibility) of variables.

JavaScript has 3 types of scope:

Block scope

Function scope

Global scope

**Blok Kapsamı**

ES6'dan (2015) önce JavaScript'te yalnızca Genel Kapsam ve İşlev Kapsamı vardı.

ES6, iki önemli yeni JavaScript anahtar sözcüğünü tanıttı: let ve const.

Bu iki anahtar kelime, JavaScript'te Blok Kapsamı sağlar.

Bir { } bloğu içinde bildirilen değişkenlere, bloğun dışından erişilemez:

{var x = 2;}  
// x CAN be used here

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

**Local Scope**

// code here can NOT use carName  
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}  
// code here can NOT use carName

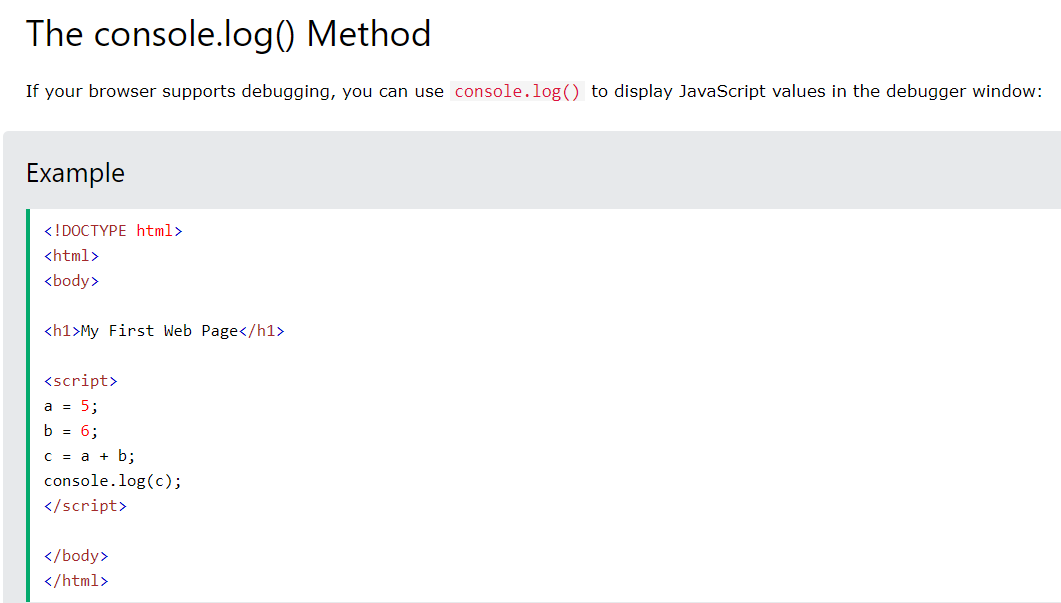
**Function Scope**

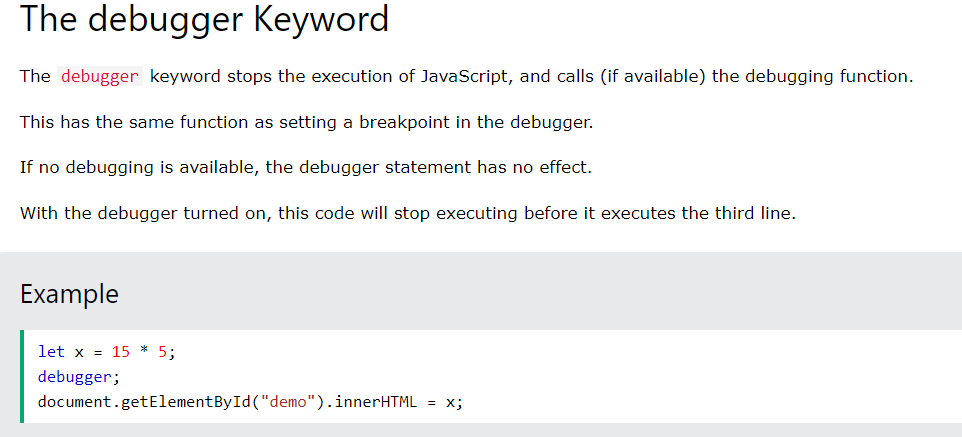
function myFunction() {  
  let carName = "Volvo";   // Function Scope  
}

**Global scope**

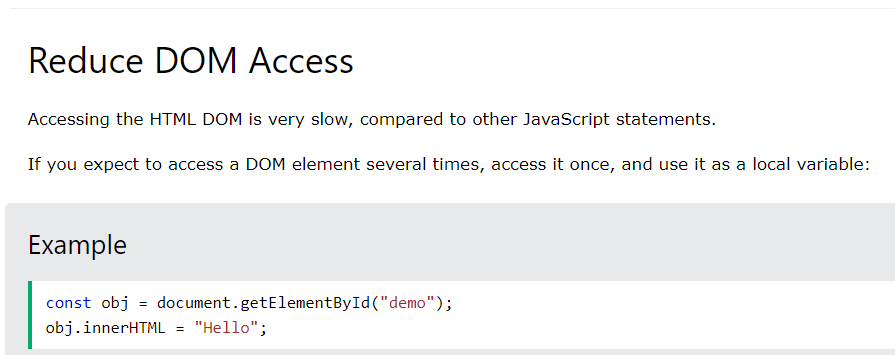
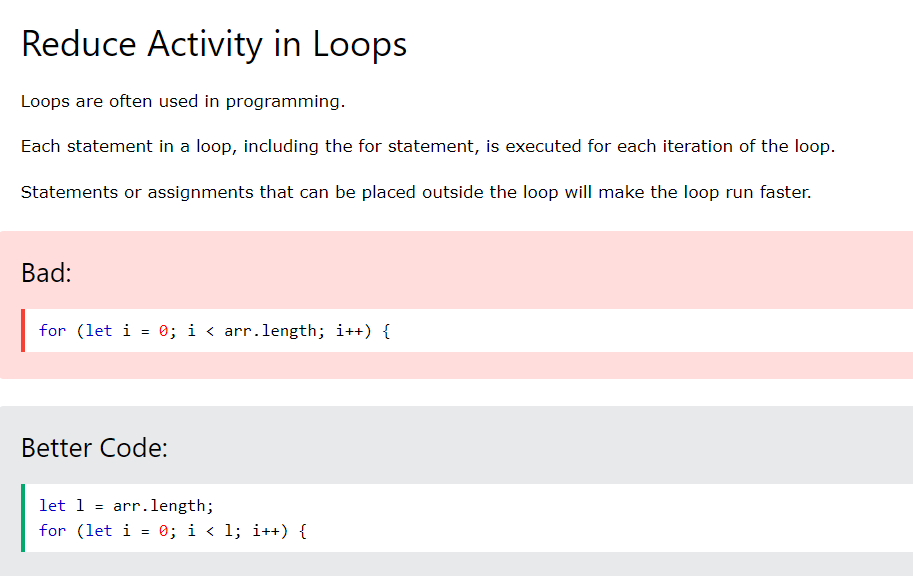
let carName = "Volvo";  
// code here can use carName  
function myFunction() {  
// code here can also use carName}

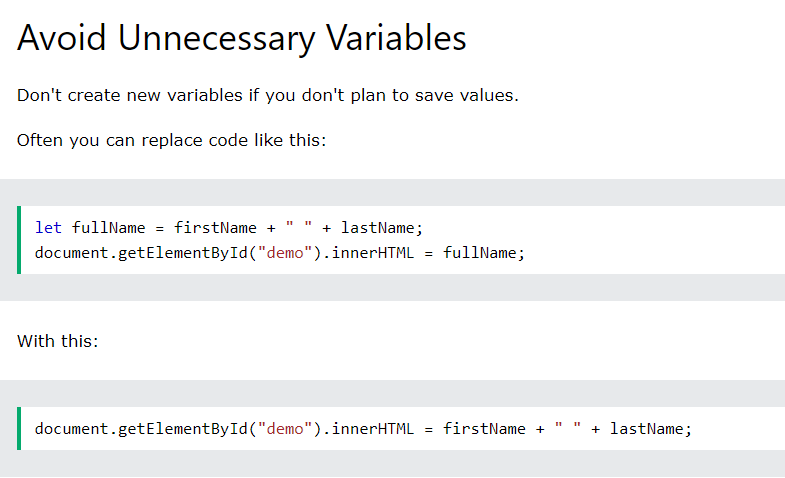
## JavaScript Debugging





## JavaScript Performance





## Advantages of JavaScript

**Hız** - JavaScript, genellikle tarayıcıda hemen çalıştırıldığı için çok hızlı olma eğilimindedir. Dış kaynaklar gerektirmediği sürece JavaScript, backend serverına yapılan çağrılar tarafından yavaşlatılmaz. Ayrıca, büyük tarayıcıların tümü JavaScripti destekler; bu, kodu çalıştırmadan önce derlemeye gerek olmadığı anlamına gelir.

**Popülerlik** - JavaScript, web'in her yerinde ve Node.js'nin ortaya çıkmasıyla birlikte, backend tarafında daha fazla kullanılmaktadır. JavaScript öğrenmek için sayısız kaynak var. Hem StackOverflow hem de GitHub, JavaScript kullanan projelerin arttığını gösteriyor.

**Birlikte Çalışabilirlik (Interoperability)** - JavaScript herhangi bir web sayfasına eklenebilir. Pearl ve PHP gibi diğer dillerdeki destek nedeniyle birçok farklı türde uygulamada kullanılabilir.

**Server Load** - JavaScript client-side tarafındadır, bu nedenle genel olarak sunuculara olan talebi azaltır ve basit uygulamaların bir sunucuya hiç ihtiyacı olmayabilir.

**Çok yönlülük (Versatility)** - JavaScript'i Node.js sunucuları aracılığıyla kullanmanın birçok yolu vardır. Node.js'yi Express ile önyükleyecekseniz, MongoDB veritabanını kullanacaksanız ve clients için frontend’de JavaScript kullanacaksanız, yalnızca JavaScript kullanarak frontend-backend uygulaması geliştirmek mümkündür.

## ES6

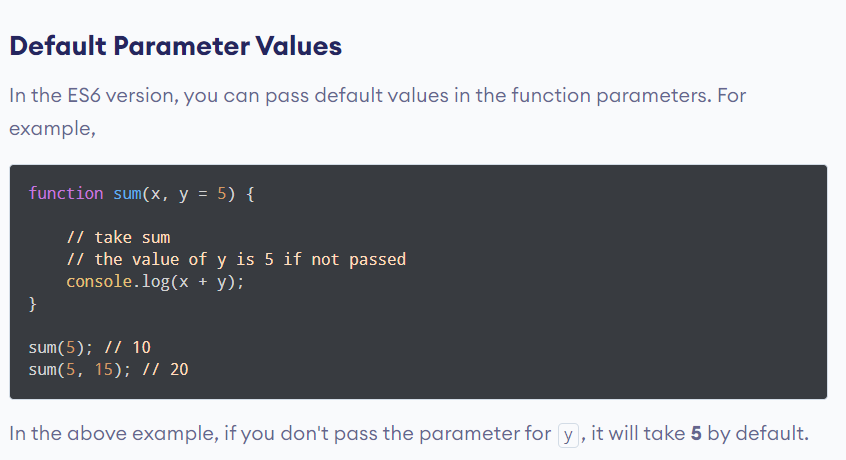
ECMAScript'in 6. baskısına ES6 adı verilir.

ES2015 olarak da bilinir.

Değişiklikler, developers’ların nesne yönelimli bir tarzda uygulamalar oluşturması sağlayan çok sayıda sözdizimsel şeker -syntactic sugar ekler.

ES6 ‘dan önce js’de proto mantığı vardı. Yani günümüz js’sinde class diye bir şey yoktur. Arka tarafta classların hepsi (ES5’e göre) functionlara dönüştürülür. Constructor mantığı ve methodlarda ES6 ile gelen bir özelliktir. Methodlar daha öncesi (ES5’e göre) prototype.methodName mantığı ile oluşturuluyordu. ES7-ES8 ve daha sonrasında gelen büyük güncellemeleri babel araçlar ile ES5’e çevrilir ve çoğu tarayıcının yeni özellikte ki kodları çalıştırmasını sağlar.

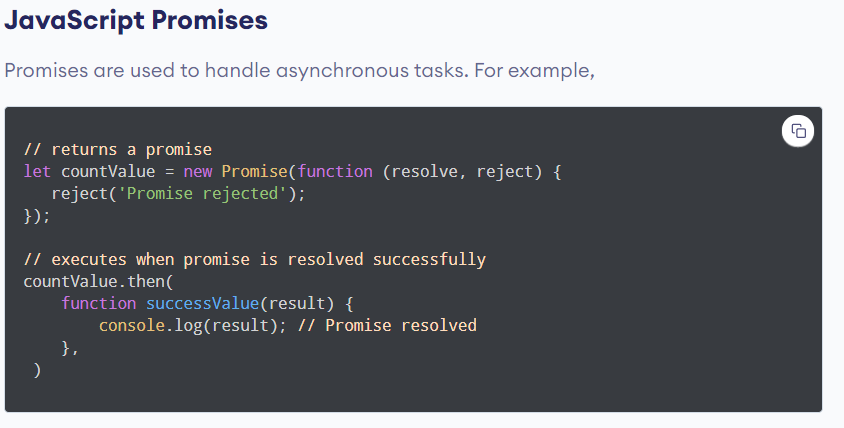




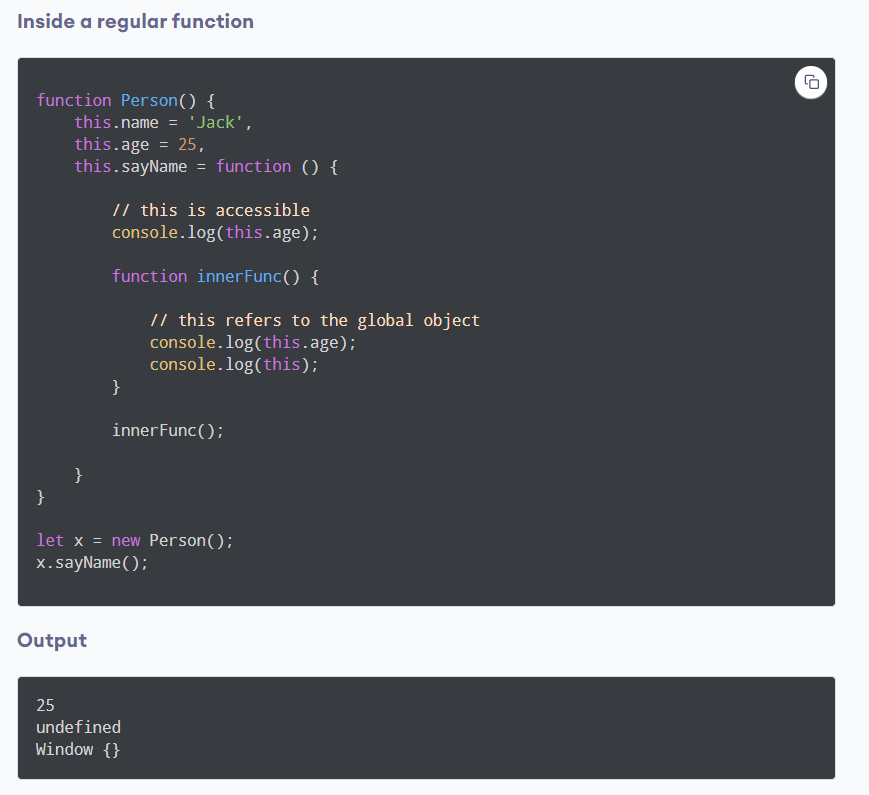




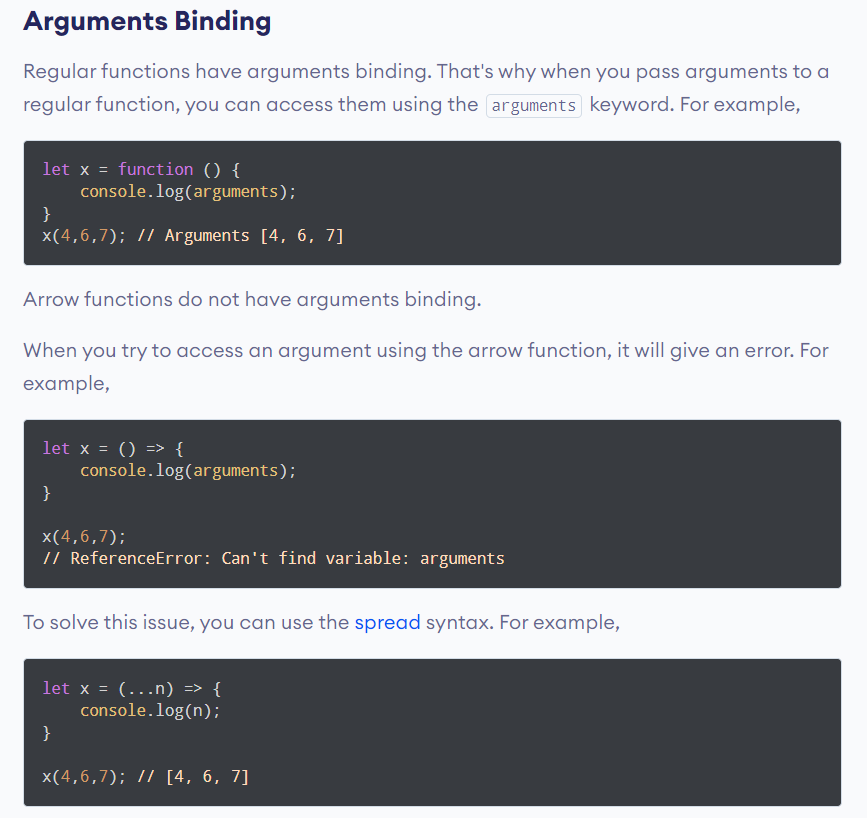


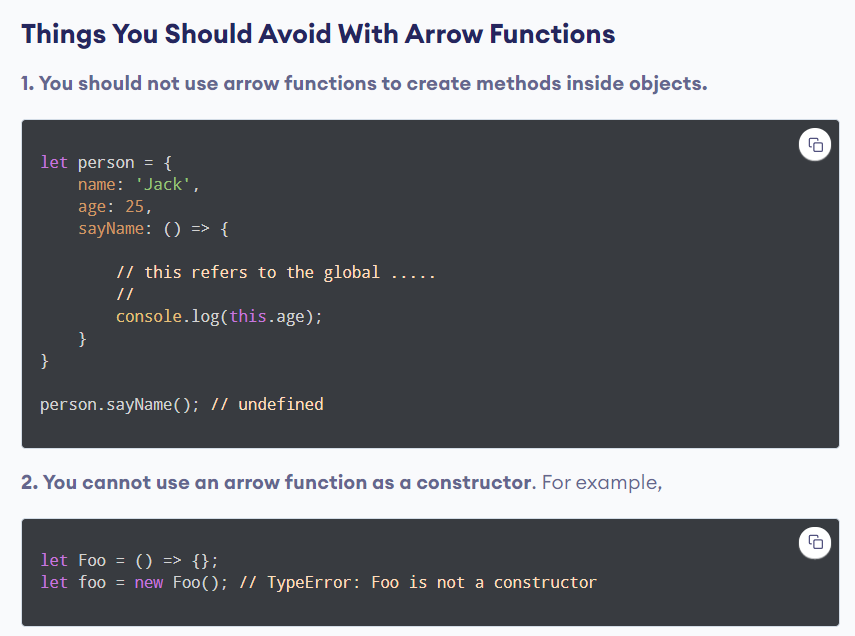
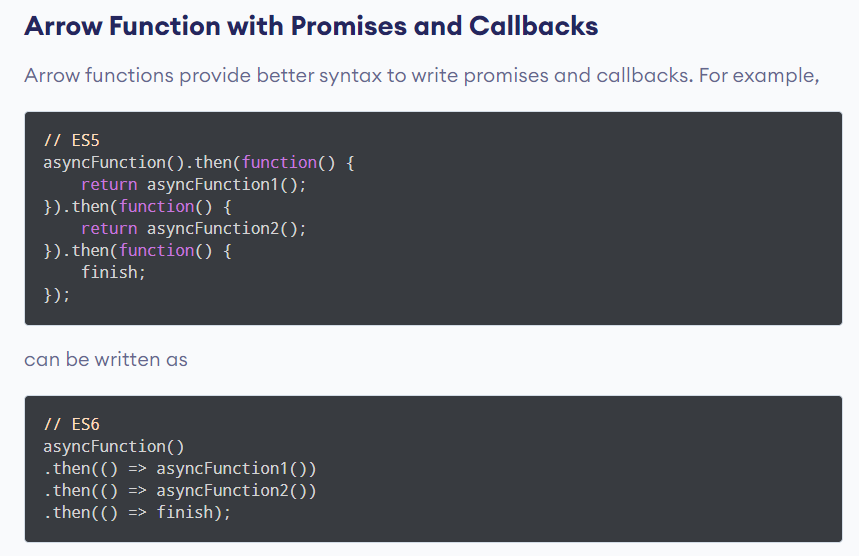


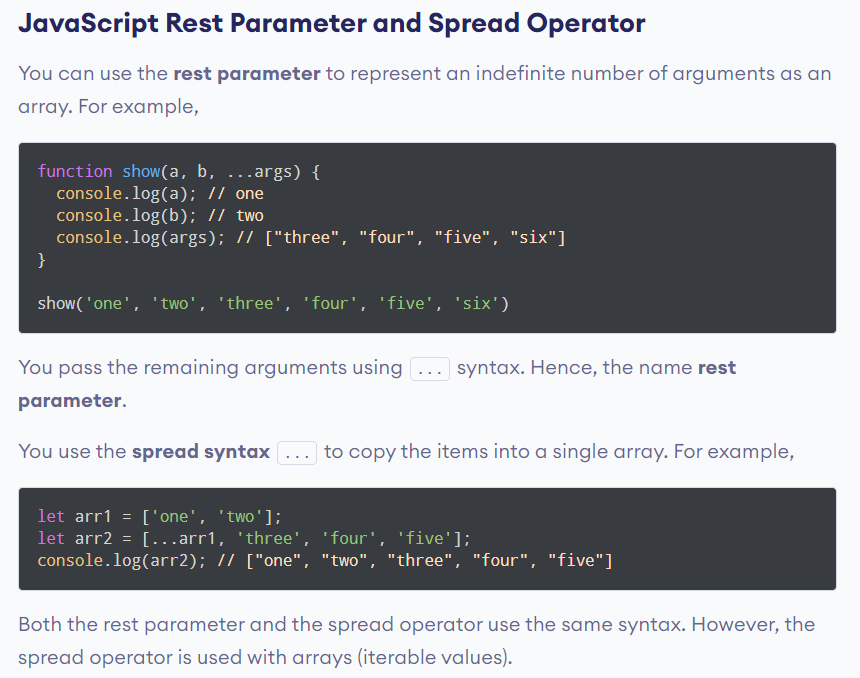
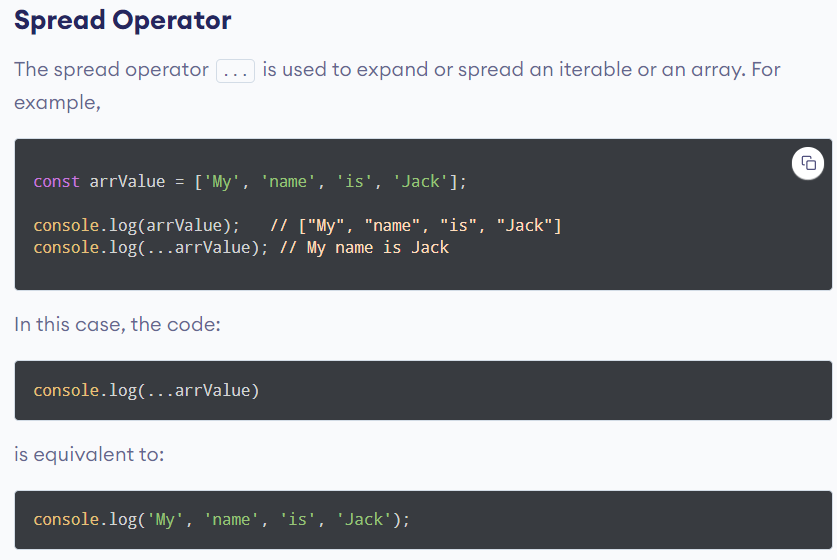


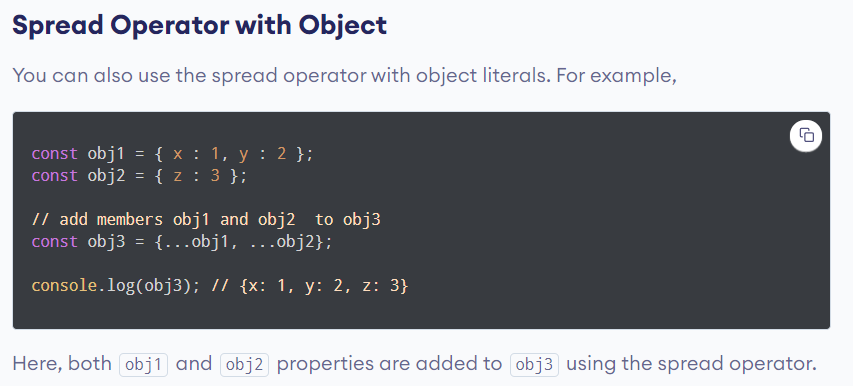


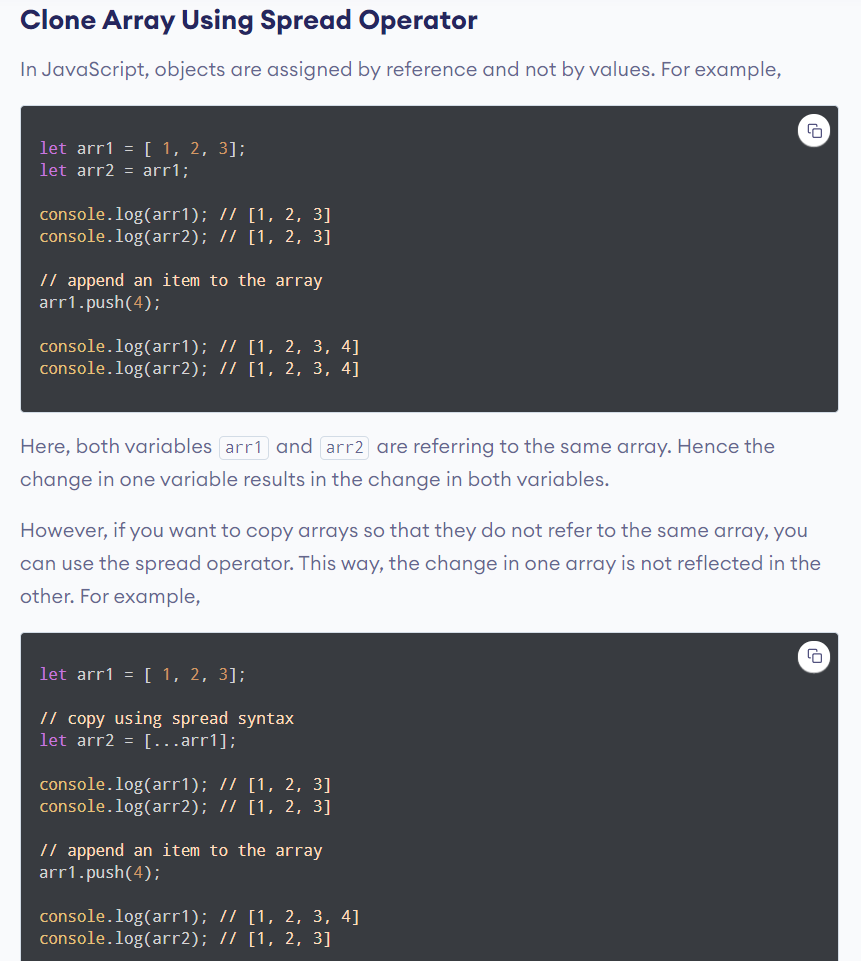


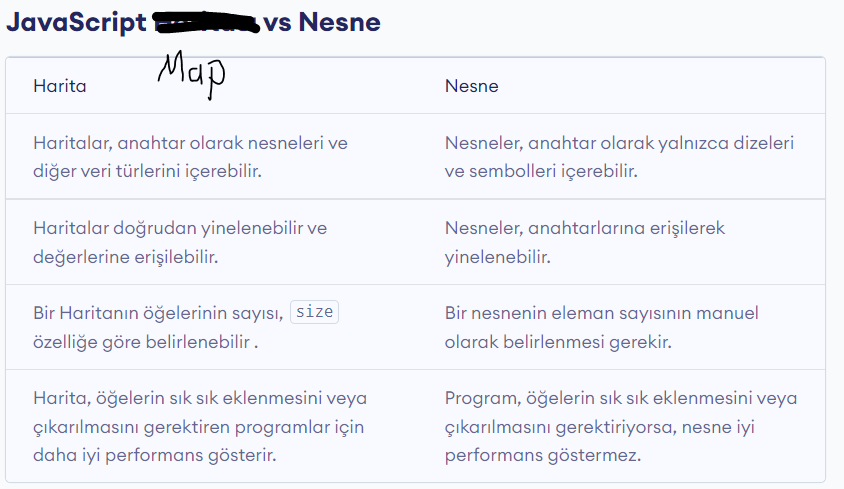
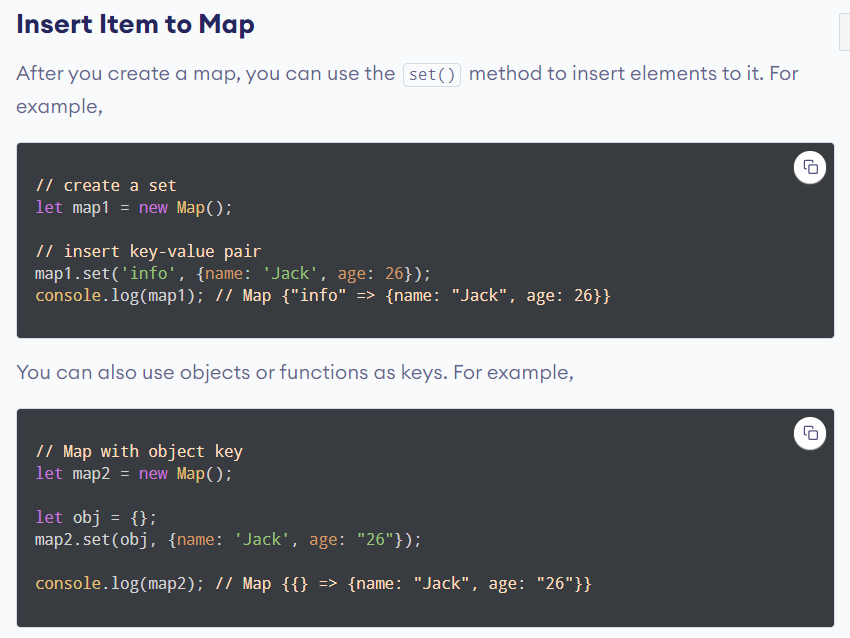


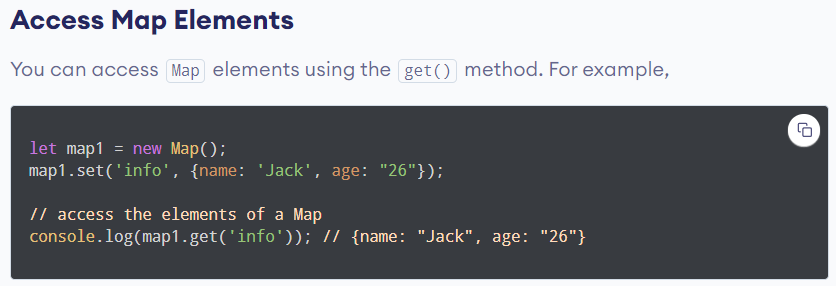




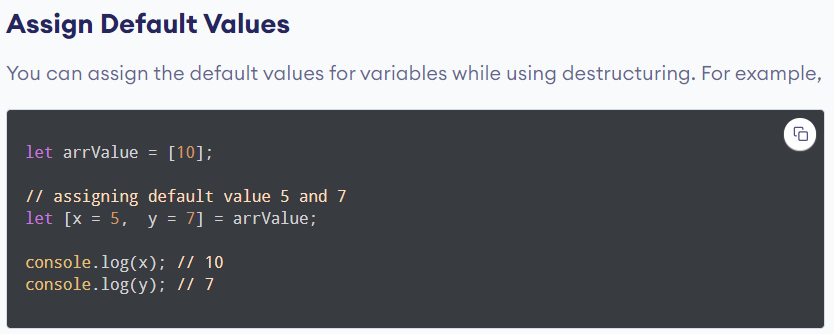
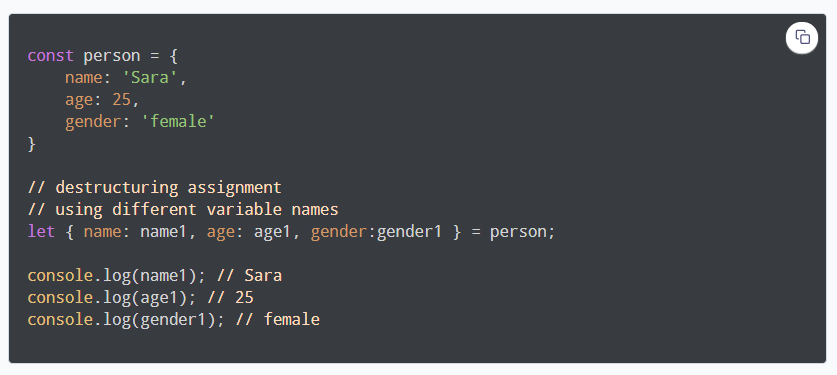


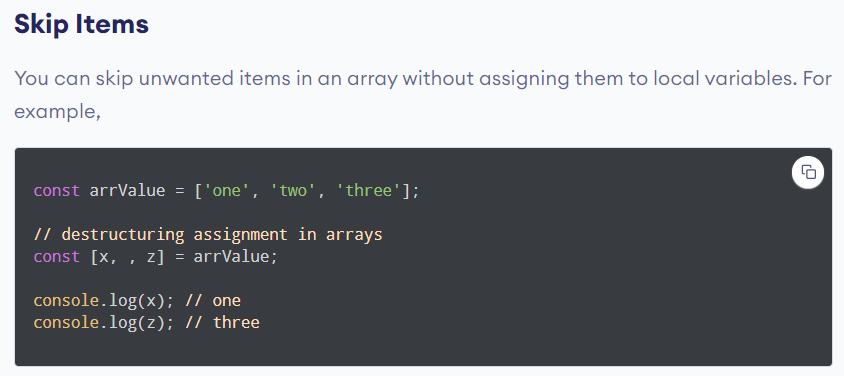
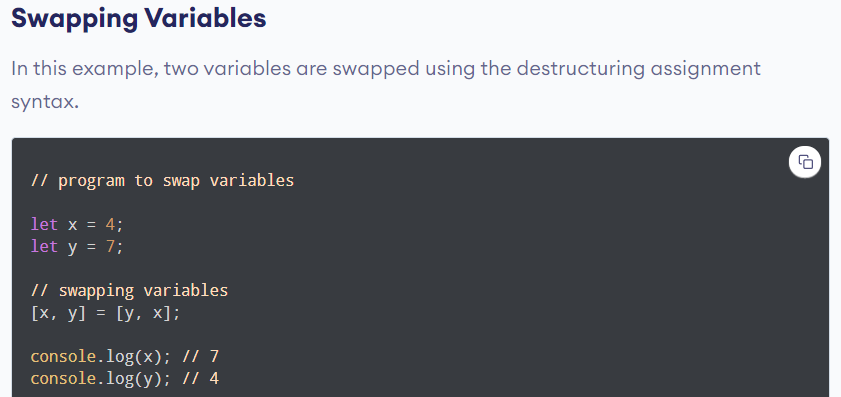


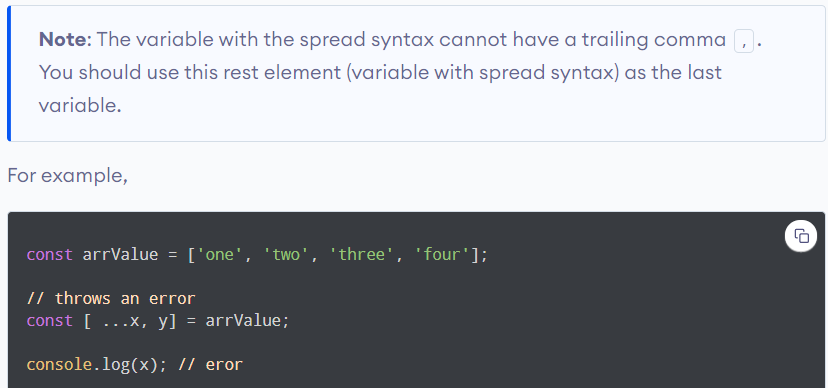


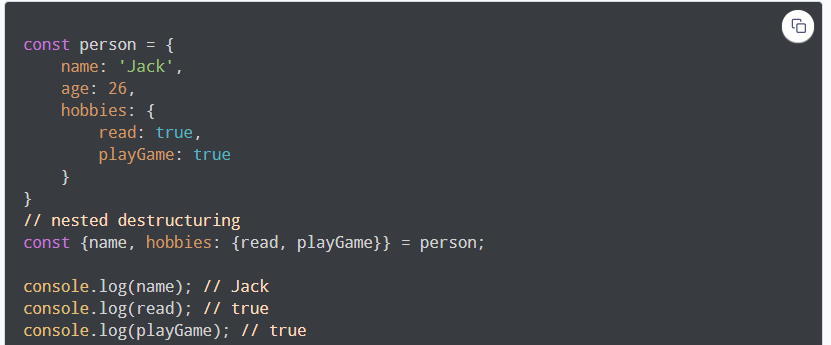
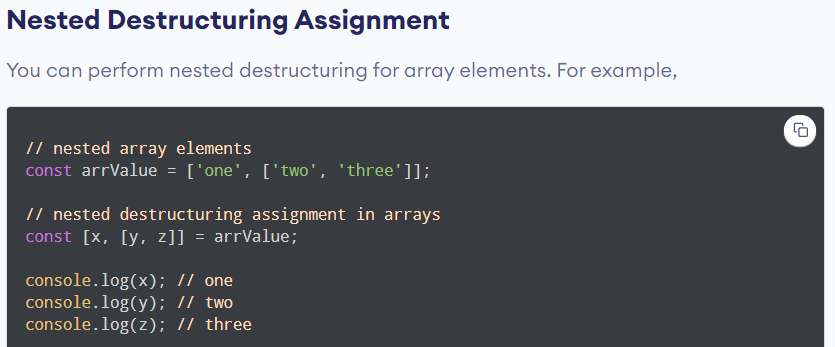


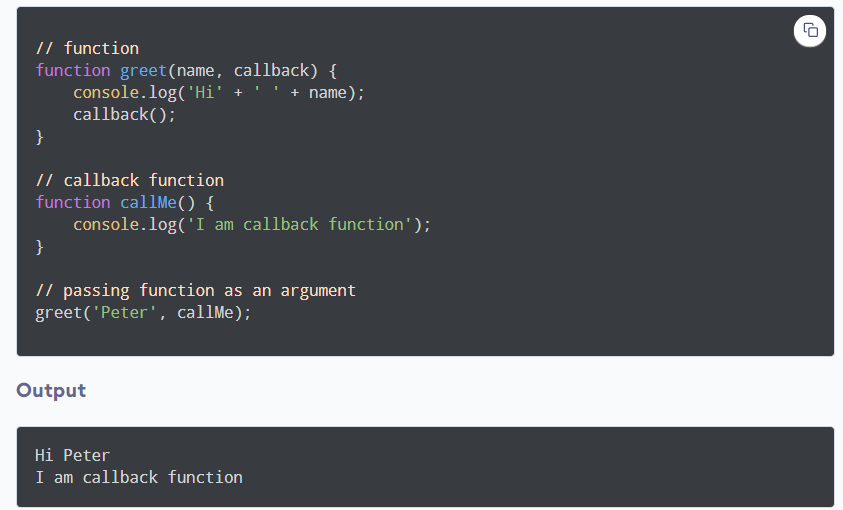
Object destructing yapıyor ve değeri de başka bir değişkene atıyor. Array dest.’de var.



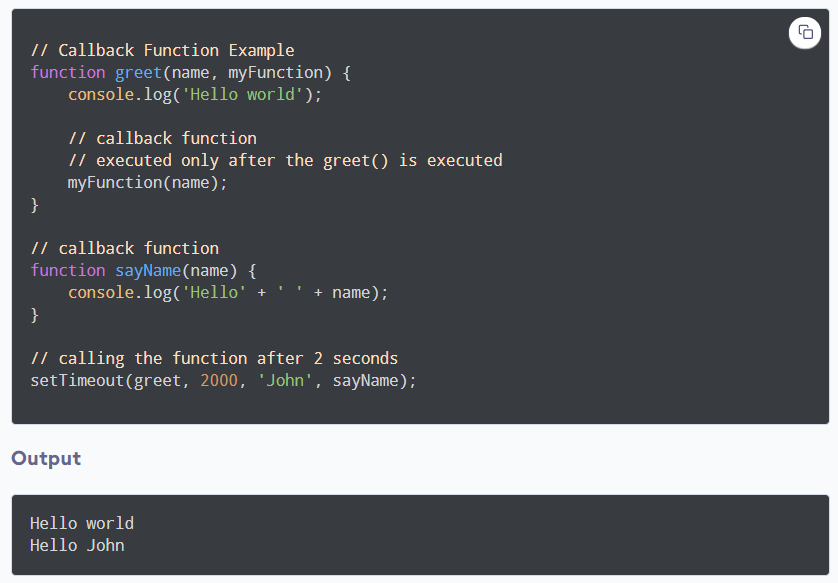












## Event-Loop Yapısı

JavaScript tek bir thread ile çalışır. Uygulamadaki tüm işleri tek bir thread koşturur. Yapılması gereken işler JavaScript Runtime’ına Event’ler ve Callback’ler ile iletilir. Tüm bu eventler ve callbackler bir kuyrukta tutulur. Thread her defasında kuyruktaki ilk eventi işler. Bu işleme Event Loop adı verilir. Node.js performanslı ve gerçek zamanlı veriye dayalı uygulamalar geliştirmek için kullanılabilecek önde gelen teknolojilerden birisidir.

**Non-Blocking Teknolojisi**

I/O ve network işlemlerini non-blocking olarak çalıştıran Node.js, zaman ve kaynak kullanımı konusunda çok başarılıdır.

Non-bloking; bir uygulama üzerinde bir işlem yaparken işlemlerin birbiririni beklemediği, asekron olarak gerçekleştiği anlamına gelir.

## Server-side ve Client-side

client-side, tarayıcıların anlayacağı taraftır. Bir de server tarafında çalıştırdığımız kodlar vardır, bunlar database işlemleri gibi işleri hallettiğimiz server-side tarafıdır. Bu iki taraf HTTP request’ler ve response’lar ile iletişim kurarlar.

## Node.js Nedir ?

Node.js, Chrome'un V8 JavaScript motoru üzerine kurulmuş, açık kaynaklı bir sunucu tarafında çalışan bir ortamıdır. JavaScript kullanarak (Asynchronous ) ortamı sağlar.

Node.js, beklemeyi ortadan kaldırır ve bir sonraki istekle devam eder. Node.js, single-threaded, non-blocking, asynchronously programming çalıştırır.

1. **Node.js Nedir?**

Node.js, JavaScript ile server side uygulamalar yazabileceğimiz, Joyent tarafından 2009 yılında geliştirilmeye başlanmış bir Javascript Runtime platformudur.

Node js, V8 isimli bir javascript motoru üzerinde çalışır.

V8, Google tarafından geliştirilen, Chrome web browserlarının da üzerinde çalıştığı C, C++ ve javascript dilleri ile kodlanan açık kaynak kodlu bir motordur.

Node.js; bir JavaScript kodunu sadece tarayıcılarda değil aynı zamanda bilgisayarınızda bağımsız şekilde çalışacak bir uygulama şeklinde kullanmak istenmesinden ortaya çıkmıştır.

Böylece JavaScript sadece web uygulamaları için kullanılan bir teknoloji olmaktan çıkmış, Python gibi Java gibi programlama dilleri ile aynı kapasitelere ulaşmıştır.

V8 engine JavaScript kodu makine koduna çevirdiği için uygulamalar çok hızlı performanslara erişebilmektedir.

## Node.js'nin Avantajları

Tüm sunucu tarafı uygulamasını oluşturmak için JavaScript kullanır.Bir uygulamanın ihtiyacına göre diğer modüller dahil edilebilir.

Varsayılan olarak Asynchronous . Bu yüzden diğer frameworklerden daha hızlı performans gösterir.