Javascript

Javascript browser tabanlı bir programlama dilidir. Dinamik web sayfaları oluşturmak için kullanılır.

|  |  |
| --- | --- |
| Javascript | |
| Tasarımcı : | Brendan Eich |
| Dosya Uzantıları : | .js |
| MIME Türü : | text/javascript |
| İlk Sürüm : | 1995 |

* Javascript kodlarının çalıştırılmadan önce derlenmesine gerek yoktur. Javascript motorları otomatikmen javascript kodlarını ayrıştırır ve makine diline dönüştürür. Kodlar belge içindeki yazılma sırasına göre yukarıdan aşağıya(ve soldan sağa) doğru çalıştırılır.
* Javascript 1997 yılında ECMA standardı olmuştur ve dilin resmi adı **ECMAScript**’tir. ECMA, yani European Computer Manufacturers Association(Avrupa Bilgisayar Üreticileri Birliği) bilgisayar sektöründe üretilen ürünler için standartlar oluşturan bir kuruluştur.

Javascript Sürümleri

|  |  |
| --- | --- |
| Version | Year |
| ECMAScript 1 | 1997 |
| ECMAScript 2 | 1998 |
| ECMAScript 3 | 1999 |
| ECMAScript 5 | 2009 |
| ECMAScript 6(ECMAScript 2015) | 2015 |
| ECMAScript 2016 | 2016 |
| ECMAScript 2017 | 2017 |
| ECMAScript 2018 | 2018 |
| ECMAScript 2019 | 2019 |
| ECMAScript 2020 | 2020 |

* 2016’dan beri sürümler yıllara göre adlandırılmaktadır.
* Javascript sürümleri hakkında detaylı bilgi için aşağıdaki web sayfasını ziyaret edebilirsiniz.

<https://www.w3schools.com/js/js_versions.asp>

Kod Yazma Notasyonları

Aşağıdaki notasyonları kullanarak kodlarınızın daha okunabilir olmasını sağlayabilirsiniz.

CamelCase, camelCase : Kelimeler bitişik ve her kelimenin ilk harfi büyük yazılır. İlk kelimenin ilk harfi küçük yazılabilir.

Snake\_Case, snake\_case : Kelimeler alt tire ile birbirine bağlanır.

SCREAMING\_SNAKE\_CASE : Genellikle sabitler için kullanılır.

kebap-case : Küçük harfler kullanılır ve kelimeler arasında tire bulunur. (Javascript bu notasyonu desteklemez)

İsimlendirme Kuralları

Javascript’te değişken ve fonksiyon isimlendirmelerinde aşağıdaki kurallar geçerlidir.

* İsimler rakam ile başlayamaz ancak rakam içerebilirler.
* \_ ve $ dışında özel karakterler kullanılamaz.
* Türkçe’ye özel karakterlerin kullanılması tavsiye edilmez.
* Anahtar kelimeler(var, function, class vb.), deyimler(continue, break vb.) ve rezerve edilmiş kelimeler isim olarak kullanılamaz.

|  |
| --- |
| **NOT**  Reserved words :  <https://www.w3schools.com/js/js_reserved.asp> |

Yorum Satırları(Comment Lines)

Javascript kodları içerisinde açıklama metinlerine yer verilmek istendiğinde aşağıdaki ifadeler kullanılmaktadır.

//

/\* \*/

Escape Characters(Kaçış Karakterleri)

\n : yeni satır.

\t : tab.

\\ : backslash.

\” : tırnak içinde tırnak.

\’ : tek tırnak içinde tek tırnak.

* \n ve \t konsol komutlarıya beraber kullanılmaktadır.
* \\, \”, \’ string ifadeler içerisinde kullanılmaktadır.

Data Types(Veri Tipleri)

Javascript’de tipler ilkel türler ve referans türler olmak üzere ikiye ayrılmaktadır.

Primitive Types(İlkel Tipler)

string, number, boolean, null, undefined

string

Metinsel veriler string türündedir. Tırnak içerisinde yazılırlar. Çift ya da tek tırnak kullanılabilir.

Example

var fullName = "John Doe";

console.log(typeof fullName); //string

number

Sayısal veriler number türündedir.

Example

var age = 28;

const PI = 3.14;

var hex = 0xFF;

var binary = 0b11111111;

console.log(age, PI, hex, binary); //28 3.14 255 255

console.log(typeof age); //number

console.log(typeof PI); //number

console.log(typeof hex); //number

console.log(typeof binary); //number

* Javascript’de özel anlamı olan NaN, Infinity ve -Infinity değerleri number türündedir.

Example

console.log(NaN); //NaN

console.log(typeof NaN); //number

console.log(Infinity); //Infinity

console.log(typeof Infinity); //number

console.log(-Infinity); //-Infinity

console.log(typeof -Infinity); //number

|  |
| --- |
| **NOT**  NaN(Not a Number) bir verinin sayı olmadığını belirten bir değerdir.  Infinity ve -Infinity ise pozitif ve negatif sayı sınırının aşıldığını belirten değerlerdir. Sınırı aşan sayılar Infinity ve -Infinity ile gösterilmektedir. |

boolean

true ve false değerleri boolean türündedir.

Example

var isUpdated = true;

console.log(typeof isUpdated); //boolean

undefined

undefined değeri undefined türündedir. Türkçesi **tanımsız** anlamına gelmektedir.

Example

var fullName = undefined;

console.log(fullName); //undefined

console.log(typeof fullName); //undefined

* Değer ataması yapılmamış değişkenler undefined olmaktadır.

Example

var number;

console.log(number); //undefined

console.log(typeof number); //undefined

null

null aslında bir tip değil bir değerdir. Boş veriler için null değeri kullanılmaktadır. Tipi object olmasına rağmen null değer içeren değişkenler ilkel tipler gibi davranmaktadır. Bu yüzden null ilkel tip olarak kabul edilmektedir.

Example

var fullName = null;

console.log(fullName); //null

console.log(typeof fullName); //object

Reference Types(Referans Tipler)

object, function

object

Objeler ve diziler object türündedir. Objeler Object sınıfından diziler ise Array sınıfından oluşturulmaktadır. Array sınıfı Object sınıfından kalıtılmıştır bu yüzden diziler object türünde olmaktadır.

Example

var obj = {};

var arr = [];

console.log(typeof obj); //object

console.log(typeof arr); //object

function

Fonksiyonlar function türündedir.

Example

function Function1(){

}

console.log(typeof Function1); //function

İlkel ve Referans Türler Arasındaki Farklar

* İlkel türlerde içerik kopyalanmaktadır.

Example

var i = 1;

var j = i;

//j, bu değişiklikten etkilenmez.

i = 2;

console.log(j); //1

Referans türlerde ise bellek adresi aktarılmaktadır.

Example

var array1 = [1, 2, 3];

var array2 = array1;

//İçerik değişikliği array2’yi etkiler. Gösterdikleri bellek bölgesi aynı.

array1.push(4);

console.log(array2); //[1, 2, 3, 4]

|  |
| --- |
| **NOT**  Eğer = operatörü ile değer ataması yeniden yapılırsa array2 bundan etkilenmez. array1 değişkeni farklı bir bellek bölgesinde yeniden tanımlanır.  Example  var array1 = [1, 2, 3];  var array2 = array1;  array1 = [4, 5, 6];  console.log(array2); //[1, 2, 3] |

* İlkel türlerde içerik, referans türlerde ise bellek adresleri kontrol edilmektedir.

Example

//Primitive Type

var i = 1;

var j = 1;

console.log(i == j); //true

Example

//Reference Type

var array1 = [1, 2, 3];

var array2 = [1, 2, 3];

console.log(array1 == array2); //false

Example

//Reference Type

var array1 = [1, 2, 3];

var array2 = array1;

console.log(array1 == array2); //true

Variables(Değişkenler)

Değişkenler bilgisayar belleğinde(RAM) veri saklamak için kullanılmaktadır.

* Javascript’de değişkenler var, let, const anahtar kelimeleri ile tanımlanmaktadır.
* Değer ataması yapılmamış değişkenler undefined olmaktadır.
* Bir değişken değer atanmaksızın yeniden tanımlanırsa değerini kaybetmez(undefined olmaz).

Example

var number = 1;

//Değer atamadan yeniden tanımlıyoruz.

var number;

console.log(number); //1

Bunun yerine undefined değeri vererek değişkenin içeriğini sıfırlayabilirsiniz.

Example

var number = 1;

number = undefined;

console.log(number); //undefined

constant Variables

const anahtar kelimesi ile sabit değişken tanımlanabilmektedir.

* İlk değer ataması zorunludur.

Example

const PI; //Yanlış

* Değer ataması yeniden yapılamaz.

Example

const name = "John";

name = "Julia"; //Yanlış.

Buna rağmen dizi ve objelerin içerikleri güncellenebilmektedir.

Example

const numbers = [1, 2, 3];

numbers.push(4); //Doğru

console.log(numbers); //[1, 2, 3, 4]

Example

const employee = {

  fullName: "John Doe",

  age: 30

}

employee.salary = 4000; //Doğru

delete employee.age; //Doğru

console.log(employee); //{fullName: "John Doe", salary: 4000}

var, let, const Farkı

* let ve const ile tanımlanan değişkenler yeniden tanımlanamazlar.

Example

var a = 1;

let b = 2;

const c = 3;

var a = 10; //Doğru

let b = 20; //Yanlış

const c = 30; //Yanlış

* Global scope’da var ile tanımlanan değişkenler window objesinin bir özelliği olurlar.

Example

//Global scope

var a = 1;

let b = 2;

const c = 3;

//Function scope

function Function1(){

  var a = 4;

  let b = 5;

  const c = 6;

  console.log(a, b, c); //4 5 6

  //Global scope'daki a değişkeni window objesinin bir özelliği oldu. this anahtar

  //kelimesi ile ulaşabilirsiniz. window objesi içerisinde b ve c adında

//değişkenler olmadığı için undefined sonucunu aldık.

  console.log(this.a, this.b, this.c); //1 undefined undefined

}

Function1();

//Fonksiyon içerisinde aynı isimde değişkenler tanımlamamıza rağmen global

//scope’daki değerler değişmedi. Çünkü function scope’da tanımlanan değişkenler

//sadece ilgili fonksiyon içerisinde yaşarlar.

console.log(a, b, c); //1 2 3

* Block scope’da var ile tanımlanan değişkenler global scope’daki değişkenleri etkileyerek istenmeyen durumlara neden olabilirler.

Example

//Global scope

var a = 1;

let b = 2;

const c = 3;

//Block scope

{

  var a = 4;

  let b = 5;

  const c = 6;

  console.log(a, b, c); //4 5 6

}

//Block scope'da var ile tanımlanan a değişkeni global scope'daki a'nın değerini

//değiştirdi.

console.log(a, b, c); //4 2 3

|  |
| --- |
| **NOT**  if-else, switch-case ve döngü blokları da block scope kapsamındadır. |

Type Conversion(Tip Dönüşümleri)

String() : Parametre olarak verilen değeri string türüne dönüştürüp geri göndermektedir.

Example

var str = String(10);

console.log(str, typeof str); //10 string

Example

var str = String(3.14);

console.log(str, typeof str); //3.14 string

Example

var str = String(true);

console.log(str, typeof str); //true string

Example

var str = String(null);

console.log(str, typeof str); //null string

Example

var str = String(undefined);

console.log(str, typeof str); //undefined string

Example

var fromArray = String([0, 1, "a", "b", true, false]);

console.log(fromArray, typeof fromArray); //0,1,a,b,true,false string

Example

function Function1(){

console.log("Hello.");

}

var str = String(Function1);

console.log(str, typeof str);

//function Function1(){ console.log("Hello."); } string

parseInt() : Parametre olarak verilen değeri number türüne dönüştürüp geri göndermektedir. Tamsayılar için kullanılır. Virgüllü sayılarda virgülden sonrası atılır.

Example

console.log(typeof parseInt("10")); //number

console.log(typeof parseInt(20)); //number

Example

const PI = parseInt(3.14); //Virgülden sonrası atılır.

console.log(PI, typeof PI); //3 number

Example

var \_null = parseInt(null);

var \_undefined = parseInt(undefined);

var \_string = parseInt("jQuery");

//NaN => Başarılı bir dönüşüm olmadı.

console.log(\_null, typeof \_null); //NaN number

console.log(\_undefined, typeof \_undefined); //NaN number

console.log(\_string, typeof \_string); //NaN number

parseFloat() , Number() : Bu fonksiyonlar parseInt()’den farklı olarak virgüllü sayıları da dönüştürebilmektedirler.

Konsol Komutları

console.log() : Konsola bilgi yazmak için kullanılır.

console.clear() : Konsolu temizlemek için kullanılır.

console.error() : Konsolda hata mesajı göstermek için kullanılır.

console.warn() : Konsolda uyarı mesajı göstermek için kullanılır.

Operators(Operatörler)

Javascript’de operatörler diğer programlama dillerinde olduğu gibi kullanılmaktadır.

|  |
| --- |
| Arithmetic Operators(Aritmetik) |
| +, -, \*, /, %, ++(Increment), --(Decrement) |

|  |
| --- |
| Assignment Operators(Atama) |
| =, +=, -=, \*=, /=, %= |

|  |
| --- |
| Comparison Operators(Karşılaştırma) |
| ==, ===, !=, !==, >, <, >=, <=, ?(Ternary Operator) |

|  |
| --- |
| Logical Operators(Mantıksal) |
| &&(Logical AND), ||(Logical OR), !(Logical NOT) |

|  |
| --- |
| Bitwise Operators(Bitsel) |
| &(AND), |(OR), ~(NOT), ^(XOR), <<(Left SHIFT), >>(Right SHIFT) |

Example

console.log(25 & 14); //8

console.log(25 | 14); //31

console.log(25 ^ 14); //23

/\*

25 : 00011001

14 : 00001110

& -> 8  : 00001000

| -> 31 : 00011111

^ -> 23 : 00010111

\*/

console.log(255 | 0); //255

console.log(255 ^ 0); //255

Example

console.log(16 << 1); //32

console.log(16 << 2); //64

console.log(16 >> 1); //8

console.log(16 >> 2); //4

console.log(17 << 1); //34

console.log(17 << 2); //68

console.log(17 >> 1); //8 -> 17/2 8.5 olmasına rağmen virgülden sonrası kayboldu.

console.log(17 >> 2); //4

Diğer Operatörler

typeof : Kendisinden sonra yazılan verinin tipini döndürür.

new : Programcı tarafından oluşturulan ya da hazır gelen sınıflardan obje üretmek için kullanılır.

this

Global scope’da this window objesini vermektedir.

Example

console.log(this); //window objesi döner.

Obje içerisinde this objenin kendisini vermektedir.

Example

let employee = {

fullName: "John Doe",

age: 30,

salary: 4000,

employeeInfo: function(){

console.log(this.fullName, this.age, this.salary); //John Doe 30 4000

}

}

employee.employeeInfo();

Fonksiyon içerisinde this window objesini verir.

Class içerisinde this ilgili class’dan oluşturulan objeyi verir.

Example

function Employee(fullName, age){

  this.fullName = fullName;

  this.age = age;

  console.log(this); //Employee objesi döner. => {fullName: "John Doe", age: 30}

}

var employee = new Employee("John Doe", 30);

delete : Bu operatör ile objelerin özellik ve metodları silinebilmektedir.

instanceof : Bu operatör ile objelerin constructor’ları öğrenilebilmektedir.

Example

var obj = {};

var arr = [];

var Class1 = function(){}

var class1 = new Class1();

console.log(obj instanceof Object); //true

console.log(obj instanceof Function); //false

console.log(arr instanceof Array); //true

console.log(arr instanceof Object); //true

console.log(arr instanceof Function); //false

console.log(class1 instanceof Class1); //true

console.log(class1 instanceof Object); //true

console.log(class1 instanceof Function); //false

Example

console.log(Function instanceof Object); //true

console.log(Object instanceof Function); //true

in : Özellik ya da metod belirtilen obje içindeyse ya da prototip zincirinde tanımlıysa true değer döndürmektedir. (Özellik, metod isimleri tırnak içinde yazılmalıdır ve büyük küçük harf ayrımına dikkat edilmelidir)

Example

console.log("pow" in Math); //true

console.log("pow()" in Math); //false

console.log("Pow" in Math); //false

Conditional Statements(Karar Yapıları)

Javascript’de karar yapıları diğer programlama dillerinde olduğu gibi kullanılmaktadır. if else ve switch case yapılarını kullanabilirsiniz.

Loops(Döngüler)

Javascript’de döngüler diğer programlama dillerinde olduğu gibi kullanılmaktadır. for, while, do while gibi standart döngüler dışında for in ve for of döngüleri mevcuttur.

for in

Example

let fruits = ["Apple", "Cherry", "Plum"];

for(var index in fruits){

  console.log(index); //0 //1 //2 => index numaraları konsola yazılır.

}

//O halde dizi içeriğine aşağıdaki gibi erişebiliriz.

for(var index in fruits){

  console.log(fruits[index]); //Apple //Cherry //Plum

}

* for in ile objeler üzerinde gezinebiliriz.

Example

var city = {

  name: "Kocaeli",

  code: 41

}

for(var key in city){

  console.log(city[key]); //Kocaeli //41

}

* for in ile dizelerin karakterlerini okuyabiliriz.

Example

let fruit = "Apple";

for(var index in fruit){

  console.log(fruit[index]); //A //p //p //l //e

}

for of

Bu döngü ECMAScript 6 ile birlikte gelmiştir.

Example

var languages = ["C#", "Python", "Java"];

for(var language of languages){

  console.log(language); //C# //Python //Java

}

Example

let fruit = "Apple";

for(var char of fruit){

  console.log(char); //A //p //p //l //e

}

|  |
| --- |
| **NOT**  for of döngüsü ile objeler üzerinde gezinme imkanımız yoktur. |

Deyimler

break : Döngüyü sonlandırmak için kullanılır.

continue : Döngü içinde belirli bir koşulun kontrolü ile bir sonraki döngü adımına gidilmesini gerçekleştirir.

return : Fonksiyonu sonlandırmak için kullanılır. return ifadesinden sonra yazılan hiçbir komut çalıştırılmaz.

İşlem Önceliği

Sırasıyla parantezler, fonksiyonlar, aritmetik işlemler, karşılaştırma işlemleri, mantıksal işlemler ve şart işlemleri.

Functions(Fonksiyonlar)

Bir Javascript fonksiyonu üç farklı yöntemle tanımlanabilmektedir.

Example

//1

function function1(){

}

//2

var function2 = function(){

}

//3.Immediately Invoked Function Expression(IIFE) : Tanımlandığı yerde çalışır.

(function(){

})()

* İç içe fonksiyon tanımlanabilir.
* Fonksiyonlar return deyimi ile geriye değer gönderebilir.

Example

function addNumbers(x, y){

return x + y;

}

function cube(x){

return x \* x \* x;

}

console.log(cube(addNumbers(1, 2))); //27

* Parametresiz fonksiyonlar parametreli olarak çağrılabilmektedir.

Example

function writeNumbers(){

  for(var index in arguments){

    console.log(arguments[index]);

  }

  console.log(arguments.length); //3

}

//Parametreler "arguments" dizisine aktarılır. Bu diziye fonksiyon içerisinden

//erişebilirsiniz.

writeNumbers(1, 2, 3);

* Fonksiyon parametrelerine default değerler verebilirsiniz.

Example

function addNumbers(x = 1, y = 2){

console.log(x + y);

}

addNumbers(); //3

addNumbers(10); //12

addNumbers(10, 20); //30

* Bir fonksiyon parantezsiz çağrılırsa fonksiyonun kendisi döner.

Example

function function1(){

console.log("buraya birşeyler yaz.");

}

var function2 = function1;

console.log(function2); //ƒ function1(){console.log("buraya birşeyler yaz.");}

function2(); //buraya birşeyler yaz.

call(), apply() ve bind() Fonksiyonları

Example

var obj1 = {number1: 10, number2: 20}

var obj2 = {number1: 30, number2: 40}

//this anahtar kelimesi normalde window objesini gösterir. call sayesinde artık

//obj1 ve obj2'yi gösterecek.

function addNumbers(number3, number4){

console.log(this.number1 + this.number2 + number3 + number4);

}

addNumbers.call(obj1, 100, 200); //330

addNumbers.call(obj2, 100, 200); //370

//apply'nin call'dan farkı parametreleri dizi değişken olarak alması.

addNumbers.apply(obj1, [100, 200]); //330

addNumbers.apply(obj2, [100, 200]); //370

//bind ise fonksiyonun bir kopyasını oluşturup parametre olarak verilen obje ile

//kullanılmasını sağlar.

var copyFunction1 = addNumbers.bind(obj1);

var copyFunction2 = addNumbers.bind(obj2);

copyFunction1(100, 200); //330

copyFunction2(100, 200); //370

Global Functions

Global fonksiyonlar window objesi altında bulunurlar.

* Tip dönüşümleri konusunda gördüğümüz String(), parseInt(), parseFloat() ve Number() fonksiyonları birer global fonksiyondur. Bu fonksiyonlar hakkında bu başlık altında tekrar bilgi verilmeyecektir.

isNaN()

Parametre olarak verilen değer sayı ise false, sayı değilse true döner.

Example

console.log(isNaN(1)); //false

console.log(isNaN("1")); //false

console.log(isNaN(NaN)); //true

console.log(isNaN(Infinity)) //false

eval()

Parametre olarak verilen string ifadeyi javascript koduymuş gibi çalıştırır.

Example

console.log(eval("2 + 2")); //4

const number = 1;

console.log(eval("number + 1")); //2

isFinite()

Bir sayının sonlu ve yasal bir sayı olup olmadığını kontrol etmek için kullanılır. Parametre olarak verilen sayı NaN, Infinity veya -Infinity ise false aksi halde true döner.

Example

console.log(isFinite(1)); //true

console.log(isFinite("1")); //true

console.log(isFinite(NaN)); //false

console.log(isFinite(Infinity)); //false

console.log(isFinite(-Infinity)); //false

Arrays(Diziler)

Bir javascript dizisi üç farklı yöntemle tanımlanabilmektedir.

Example

//1.Condensed Array

let fruits = new Array("Apple", "Cherry", "Orange");

//2.Literal Array

let numbers = [1, 2, 3];

//3.Regular Array

let days = new Array();

days[0] = "Sunday";

days[1] = "Monday";

//Hata meydana gelmez. 2, 3 ve 4. index'ler boş olarak tanımlanır. Bu yüzden dizi

//6 elemanlı olur.

days[5] = "Friday";

console.log(days); //[ "Sunday", "Monday", <3 empty slots>, "Friday" ]

console.log(days.length); //6

* Çok boyutlu diziler tanımlayabilirsiniz.

Example

let arr1 = Array(new Array(), new Array(), new Array());

let arr2 = [[], [], []];

let arr3 = new Array();

arr3[0] = new Array();

arr3[1] = new Array();

arr3[2] = new Array();

Array Sınıfı

Diziler üzerinde işlemler yapılırken Array sınıfının özellik ve metodları kullanılmaktadır.

Example

//Özellik ve metodlar konsol penceresinde görüntülenebilmektedir.

console.log(new Array());

length

Dizinin eleman sayısını almak ya da değiştirmek için kullanılır.

Example

let numbers = [1, 2, 3];

console.log(numbers); //[ 1, 2, 3 ]

console.log(numbers.length); //3

//Dizinin eleman sayısını değiştirdik. Bazı elemanlar silindi.

numbers.length = 2;

console.log(numbers); //[ 1, 2 ]

console.log(numbers.length); //2

push()

Dizinin sonuna eleman eklemek için kullanılır.

Example

let numbers = [1, 2, 3];

numbers.push(4, 5, 6);

console.log(numbers); //[ 1, 2, 3, 4, 5, 6 ]

unshift()

Dizinin başına eleman eklemek için kullanılır. Kullanımı push() metodu ile aynıdır.

pop()

Dizinin son elemanını verir ve bu elemanı dizi içinden siler.

Example

let numbers = [1, 2, 3];

console.log(numbers.pop()); //3

console.log(numbers); //[ 1, 2 ]

shift()

Dizinin ilk elemanını verir ve bu elemanı dizi içinden siler. Kullanımı pop() metodu ile aynıdır.

toString()

Dizi içeriğini string olarak döndürür.

Example

let numbers = [1, 2, 3];

let str = numbers.toString();

console.log(str); //1,2,3

console.log(typeof str); //string

join()

Parametre olarak verilen değeri dizi elemanları arasına ekler ve string olarak döndürür. Parametresiz kullanımda elemanlar arasına virgül eklenir.

Example

let numbers = [1, 2, 3];

console.log(numbers.join()); //1,2,3

console.log(numbers.join("+")); //1+2+3

console.log(numbers.join("--")); //1--2--3

concat()

Dizileri birleştirip geri döndürür.

Example

let arr1 = [1, 2, 3];

let arr2 = [4, 5, 6];

let arr3 = [7, 8, 9];

console.log(arr1.concat(arr2, arr3)); //[ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]

reverse()

Dizi elemanlarını tersten sıralar.

Example

let numbers = [1, 2, 3];

numbers.reverse();

console.log(numbers); //[ 3, 2, 1 ]

indexOf()

Parametre olarak verilen değeri dizi içerisinde arar ve ilk geçtiği yerin index numarasını döndürür. Eşleşme olmazsa -1 değeri döner. İkinci parametrenin kullanımı isteğe bağlıdır, başlangıç index’ini belirtmek için kullanılır.

Example

let numbers = [1, 9, 7, 3, 1];

console.log(numbers.indexOf(1)); //0

console.log(numbers.indexOf(2)); //-1

console.log(numbers.indexOf(1, 1)); //4

lastIndexOf()

Parametre olarak verilen değeri dizi içerisinde arar ve son geçtiği yerin index numarasını döndürür. Eşleşme olmazsa -1 değeri döner. İkinci parametrenin kullanımı isteğe bağlıdır, bitiş index’ini belirtmek için kullanılır.

Example

let numbers = [1, 9, 7, 3, 1];

console.log(numbers.lastIndexOf(1)); //4

console.log(numbers.lastIndexOf(1, 3)); //0

console.log(numbers.lastIndexOf(1, 4)); //4

slice()

Dizinin bir bölümünü(ya da tamamını) kopyalar ve yeni bir dizi olarak döndürür. İlk parametre başlangıç, ikinci parametre bitiş index’idir. Bitiş index’i belirtilmezse dizinin sonuna kadar seçilir. Bitiş index’i negatif değer aldığında dizi sonundan bitiş index’i kadar eleman seçilmez.

Example

let numbers = [1, 2, 3, 4, 5, 6];

console.log(numbers.slice()); //[1, 2, 3, 4, 5, 6]

console.log(numbers.slice(1)); //[2, 3, 4, 5, 6]

console.log(numbers.slice(1, 4)); //[2, 3, 4]

console.log(numbers.slice(0, -2)); //[1, 2, 3, 4]

splice()

Dizi içinden eleman silmek ya da yeni elemanlar eklemek için kullanılır.

Example

//Elemanları silerken yeni elemanlar eklemek.

let arr = [1, 2, 3, 4, 5, 6];

//splice(startIndex, count, newItems)

//splice(başlangıç indeksi, çıkarılacak eleman sayısı, eklenecek elemanlar)

let deletedItems = arr.splice(2, 2, "A", "B", "C");

console.log(deletedItems); //[3, 4]

console.log(arr); //[1, 2, "A", "B", "C", 5, 6]

Example

//Eleman eklemek.

let arr = [1, 2, 3, 4];

arr.splice(2, 0, "A", "B", "C");

console.log(arr); //[1, 2, "A", "B", "C", 3, 4]

Example

//Eleman silmek.

let arr = [1, 2, 3, 4, 5, 6];

arr.splice(2, 2);

console.log(arr); //[ 1, 2, 5, 6 ]

map()

Dizi elemanlarını belirtilen fonksiyondan geçirerek dizi halinde geri döndürür.

Example

let numbers = [2, 4, 6, 8];

console.log(numbers.map(function(item){ //[ 4, 16, 36, 64 ]

return item \* item;

}));

//İkinci ve üçüncü parametrelerin kullanımı isteğe bağlıdır.

console.log(numbers.map(function(item, index, arr){ //[ 4, 16, 36, 64 ]

return arr[index] \* item;

}));

//Fonksiyon bloğundan sonra kendi parametrenizi gönderebilirsiniz.

//Parametrenize fonksiyon içerisinden this operatörü ile ulaşabilirsiniz.

//NOT : Parametre gönderilmezse this undefined olur.

console.log(numbers.map(function(item){ //[ "4-", "16-", "36-", "64-" ]

return item \* item + this;

}, "-"));

filter()

Dizi elemanlarını filtreleyip dizi halinde geri döndürür.

Example

let numbers = [2, 4, 6, 8];

console.log(numbers.filter(function(item){ //[ 2, 4 ]

return item < 5;

}));

every()

Dizi elemanlarını belirli bir kontrolden geçirir, tüm elemanlar koşulu sağlıyorsa true aksi halde false döner.

Example

let numbers = [2, 4, 6, 8];

console.log(numbers.every(function(item){ //true

return item <= 8;

}));

console.log(numbers.every(function(item){ //false

return item <= 6;

}));

some()

every() gibi çalışır fakat elemanlardan en az biri koşulu sağlıyorsa true hiçbir eleman koşulu sağlamıyorsa false döner.

Example

let numbers = [2, 4, 6, 8];

console.log(numbers.some(function(item){ //true

return item <= 2;

}));

console.log(numbers.some(function(item){ //false

return item > 8;

}));

forEach()

Dizideki her eleman için bir fonksiyon çalıştırır.

Example

let numbers = [2, 4, 6, 8];

numbers.forEach(function(item){

console.log(item); //2 //4 //6 //8

});

|  |
| --- |
| **NOT**  filter(), some(), every() ve forEach() metodları da map() gibi iki ya da üç parametreli olarak kullanılabilmektedir. Ayrıca fonksiyon bloğundan sonra özel parametre gönderme imkanımız vardır. |

Error Management(Hata Yönetimi)

Javascript’te hata yönetimi try, catch, finally blokları ile yapılmaktadır.

Example

try{

  //Bu blok kodları içerir.

}

catch(err){

  //Hata meydana gelirse catch bloğu çalışır.

  //Browser tarafından bir Error objesi fırlatılır. Bu obje catch bloğuna verilen

  //parametre(err) ile yakalanır.

  //Error objesi name ve message adında iki özelliğe sahiptir. Bu özellikler bize

  //oluşan hata hakkında bilgi vermektedir.

}

finally{

  //Hata olsun olmasın çalışması istenen kodlar buraya yazılır.

}

Example

try{

//(y adında bir değişken yok). Bu kod satırı try bloğu içerisinde olmasaydı

//Error objesi browser tarafından yakalanıp konsola yazılırdı. Ama artık bizim

//kontrolümüzde. Browser bu hatadan etkilenmeyecek.

var x = y + 1;

}

catch(err){

console.log(err); //ReferenceError: y is not defined

console.log(err.name); //ReferenceError

console.log(err.message); //y is not defined

}

* Bir Error objesinin name özelliği gerçekleşen hataya göre şu beş farklı değerden birini alabilir : RangeError, ReferenceError, SyntaxError, TypeError, URIError
* Error objesinin message özelliği gerçekleşen hatanın açıklamasını gösteren bir string’dir. Açıklama metni browser’a göre değişir.
* Kendi Error objelerinizi oluşturabilir ve bunları throw anahtar kelimesi ile fırlatabilirsiniz.

Example

var myError = new Error();

myError.name = "Error Name";

myError.message = "Error Message";

try{

throw myError;

}

catch(err){

console.log(err); //Error Name: Error Message

console.log(err.name); //Error Name

console.log(err.message); //Error Message

}

* throw anahtar kelimesi ile fırlattığınız hata string, number, boolean veya object türünde olabilir.

OOP(Object Oriented Programming)

Javascript’te obje oluşturmanın farklı yolları vardır. En basit yöntem **object literal** sözdizimini kullanmaktır.

Example

let employee = {

fullName: "John Doe",

age: 30,

salary: 4000

}

console.log(employee); //{fullName: "John Doe", age: 30, salary: 4000}

Obje oluşturmanın başka bir yolu ise Object sınıfını kullanmaktır. Bu yöntemin object literal sözdiziminden hiçbir farkı bulunmamaktadır.

Example

let employee = new Object();

employee.fullName = "John Doe";

employee.age = 30;

employee.salary = 4000;

console.log(employee); //{fullName: "John Doe", age: 30, salary: 4000}

Javascript’de fonksiyonlar üzerinden de obje oluşturulabilmektedir.

Example

function Person(firstName, age){

  this.firstName = firstName;

  this.age = age;

}

var person = new Person("John", 30);

console.log(person); //Person {firstName: 'John', age: 30}

class anahtar kelimesini kullanarak da obje oluşturabilmekteyiz.

Example

class Person{

  constructor(firstName, age){

    this.firstName = firstName;

    this.age = age;

  }

}

var person = new Person("John", 30);

console.log(person); //Person {firstName: 'John', age: 30}

* Javascript’te tüm sınıflar Object sınıfının mirasçısıdır.
* Tüm objelerin prototype ve constructor özellikleri bulunmaktadır. prototype özelliği objenin prototipini, constructor özelliği ise objenin constructor’ını döndürmektedir.

Example

var obj1 = {}

var obj2 = new Object();

function Class1(){}

var obj3 = new Class1();

console.log(obj1.\_\_proto\_\_); //Object sınıfının prototipi döner.

console.log(obj2.\_\_proto\_\_); //Object sınıfının prototipi döner.

console.log(obj3.\_\_proto\_\_); //Objenin kendi prototipi döner.

Example

var obj1 = {}

var obj2 = new Object();

function Class1(){}

var obj3 = new Class1();

//obj1 ve obj2 Object sınıfının constructor'ını kullanır.

//obj3 için biz bir constructor tanımladık(Class1).

console.log(obj1.constructor); //ƒ Object() { [native code] }

console.log(obj2.constructor); //ƒ Object() { [native code] }

console.log(obj3.constructor); //ƒ Class1(){}

* Objelerimizin prototipleri içerisinde başlangıçta herhangi bir özellik ve metod bulunmamaktadır. Object sınıfının prototipi ise geliştiriciler tarafından tanımlanan bazı özellik ve metodları içermektedir. Objelerimizin prototipleri Object sınıfının prototipini de içerisinde barındırmaktadır. Çünkü Javascript’te tüm sınıflar Object sınıfının mirasçısıdır. Object sınıfının prototipi içerisinde bulunan özellik ve metodları objelerimiz üzerinde kullanabiliriz. Hatta bu özellik ve metodları objelerimiz içerisinde override edebiliriz.

Aşağıdaki örnekte object sınıfı kaynaklı prototip içerisinde yer alan toString() metodunun nasıl override edildiği gösterilmektedir.

Example

function Person(firstName, age){

  this.firstName = firstName;

  this.age = age;

  this.toString = function(){

    console.log("toString Method");

  }

}

var person = new Person("John", 30);

person.toString(); //toString Method

//obje içerisi > objenin prototipi > Object sınıfının prototipi.

//Yukarıdaki sıralamaya göre toString() metodu ilk tanımlandığı yerde çalışacak.

* Bir objenin prototipi içerisinde özellik veya metod tanımlayabiliriz.

Example

function Person(firstName, age){

  this.firstName = firstName;

  this.age = age;

}

Person.prototype.showInfos = function(){

  console.log(this.firstName, this.age);

}

var person = new Person("John", 30);

person.showInfos(); //John 30

console.log(person); //Person {firstName: 'John', age: 30}

Yukarıdaki metod doğrudan objenin içerisinde tanımlansaydı her obje için bellekte bir kopyası olurdu. Prototip içerisinde tanımladığımız için bellekte sadece bir adet metod yer alacak.

* Bir objenin prototipini başka bir objenin içeriğinden oluşturabiliriz.

Example

var obj1 = {

firstName: "John"

}

//Boş bir obje oluşturur ve bu objenin prototipi içerisinde firstName özelliği //olur.

console.log(Object.create(obj1));

Example

var obj1 = {

firstName: "John"

}

var obj2 = Object.create(obj1);

console.log(obj2);

* Bir objenin prototipi başka bir objenin prototipini gösterebilir. Bu sayede kalıtım yapabiliyoruz.

Example

function Person(firstName, age){

  this.firstName = firstName;

  this.age = age;

}

Person.prototype.showInfos = function(){

  console.log(this.firstName, this.age);

}

function Employee(firstName, age){

  this.firstName = firstName;

  this.age = age;

}

//Count özelliği silinecek.

Employee.prototype.Count = 0;

Employee.prototype = Person.prototype;

var employee = new Employee("John", 30);

employee.showInfos(); //John 30

console.log(employee); //Employee {firstName: 'John', age: 30}

Yukarıdaki örnekte employee objesinin kendi prototipi olmaz. Çünkü kendi prototipi Person sınıfının prototipini göstermektedir. employee objesinin kendi prototipi olsun ve içerisinde Person sınıfının prototipi bulunsun isteseydik

Employee.prototype = Person.prototype;

yerine

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

kullanırdık.

Konuyu fonksiyon tabanlı basit bir kalıtım örneği ile bitirebiliriz.

Example

function Person(firstName, age){

  this.firstName = firstName;

  this.age = age;

}

Person.prototype.showInfos = function(){

  console.log(this.firstName, this.age);

}

function Employee(firstName, age, salary){

  Person.call(this, firstName, age);

  this.salary = salary;

}

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

Employee.prototype.showInfos = function(){

  console.log(this.firstName, this.age, this.salary);

}

var employee = new Employee("John", 30, 4000);

employee.showInfos(); //John 30 4000

console.log(employee); //Employee {firstName: 'John', age: 30, salary: 4000}

Javascript Objeleri

String

prototype

constructor

length

String ifadenin uzunluğunu verir.

Example

console.log("John".length); //4

charAt(index)

String ifade içinden belirtilen konumdaki karakteri verir. İlgili index yoksa boş string döner.

Example

let str = "John";

console.log(str.charAt(str.length - 1)); //n

charCodeAt(index)

String ifade içinden belirtilen konumdaki karakterin ASCII kodunu verir. (Büyük, küçük harf duyarlılığı vardır)

Example

console.log("Apple".charCodeAt(0)); //65

concat()

String ifadeleri birleştirip geri döndürür.

Example

console.log("John".concat("Doe")) //JohnDoe

String.fromCharCode()

Girilen ASCII değerinin karakterini döndürür.

Example

console.log(String.fromCharCode(84, 82, 65, 75, 89, 65)); //TRAKYA

indexOf()

Parametre olarak verilen değeri string ifade içinde arar. Bulursa ilk geçtiği yerin index numarasını bulamazsa -1 döndürür. İkinci parametrenin kullanımı isteğe bağlıdır, başlangıç index’ini belirtmek için kullanılmaktadır.

lastIndexOf()

Parametre olarak verilen değeri string ifade içinde arar. Bulursa son geçtiği yerin index numarasını bulamazsa -1 döndürür. İkinci parametrenin kullanımı isteğe bağlıdır, bitiş index’ini belirtmek için kullanılmaktadır.

replace()

String ifade içinde değişiklik yapıp geri döndürür.

Example

let str = "aa aa";

console.log(str.replace("aa", "bb")); //bb aa -> Değişiklik sadece ilk "aa" için yapılır.

console.log(str); //aa aa -> Orjinal veri değişiklikten etkilenmez.

toLowerCase(), toUpperCase()

Example

console.log("IĞDIR".toLowerCase()); //iğdir -> I harfi i'ye dönüştü.

console.log("izmir".toUpperCase()); //IZMIR -> i harfi I'ya dönüştü.

toLocaleLowerCase(), toLocaleUpperCase()

Tarayıcının diline göre büyük-küçük harf dönüşümü yapmaktadır.

Example

console.log("IĞDIR".toLocaleLowerCase()); //ığdır

console.log("izmir".toLocaleUpperCase()); //İZMİR

substring()

String ifade içinden belirtilen iki konum arasındaki karakterleri getirir.

Example

let password = "QWERTYUIOP";

console.log(password.substring(0, 5)); //QWERT

console.log(password.substring(2)); //ERTYUIOP

split()

String ifadeyi bölerek dizi olarak döndürür. İlk parametre ayırıcı karakterdir, ikinci parametre ile dizinin eleman sayısı ayarlanabilmektedir.

Example

let webSite = "www.yazilimblog.com";

console.log(webSite.split(".")); //['www', 'yazilimblog', 'com']

console.log(webSite.split(".", 2)); //['www', 'yazilimblog']

toString()

String ifadeyi döndürür.

Number

prototype

constructor

MAX\_VALUE

Mümkün olan en büyük sayıyı döndürür. MAX\_VALUE’dan büyük sayılar Infinity ile gösterilmektedir.

MIN\_VALUE

Mümkün olan en küçük sayıyı döndürür. MIN\_VALUE’dan küçük sayılar -Infinity ile gösterilmektedir.

NaN

toFixed()

Virgülden sonraki basamak sayısını ayarlayıp string olarak geri döndürür.

Example

let number = 123.456789;

console.log(number.toFixed(1)); //123.5

console.log(number.toFixed(2)); //123.46

console.log(number.toFixed(3)); //123.457

toString()

Sayıyı string olarak döndürür. Parametre kullanımı isteğe bağlıdır, sayı tabanını ayarlamak için kullanılır. Parametre 2 ile 36 arasında bir tamsayı olmalıdır.

Example

var number = 100;

console.log(number.toString()); //100 -> Taban 10

console.log(number.toString(2)); //1100100 -> Binary

console.log(number.toString(8)); //144 -> Octal

console.log(number.toString(16)); //64 -> Hexadecimal

Error

name

message

Boolean

prototype

constructor

toString()

Math

Math objesinin constructor’ı yoktur. new anahtar kelimesi ile obje oluşturulamaz.

E

Eular sayısını döndürür.

Example

console.log(Math.E); //2.718281828459045

PI

PI sayısını döndürür.

Example

console.log(Math.PI); //3.141592653589793

LN2

2’nin logaritmasını döndürür.

LN10

10’un logaritmasını döndürür.

SQRT1\_2

1/2’nin karekökünü döndürür.

SQRT2

2’nin karekökünü döndürür.

abs()

Parametre olarak verilen sayının mutlak değerini döndürür.

Example

console.log(Math.abs(-5)); //5

console.log(Math.abs(-5/10)); //0.5

console.log(Math.abs(-0.2)); //0.2

sqrt()

Parametre olarak verilen sayının karekökünü döndürür.

Example

console.log(Math.sqrt(36)); //6

console.log(Math.sqrt(2.25)); //1.5

console.log(Math.sqrt(10\*3.6)); //6

pow()

Üs almak için kullanılır.

Example

console.log(Math.pow(3, 2)); //9

console.log(Math.pow(3, 3)); //27

console.log(Math.pow(3/2, 2)); //2.25

ceil()

Virgüllü sayıyı yukarı yuvarlar.

Example

console.log(Math.ceil(-2.1)); //-2

console.log(Math.ceil(8.669)); //9

console.log(Math.ceil(6)); //6

floor()

Virgüllü sayıyı aşağı yuvarlar.

Example

console.log(Math.floor(6.9)); //6

console.log(Math.floor(-6.1)); //-7

console.log(Math.floor(4.999)); //4

round()

Virgüllü sayıyı yakın olan tarafa yuvarlar. 0.5 gibi bir sonraki ve bir önceki tamsayıya eşit uzaklıktaki değerler yukarı yuvarlanır.

Example

console.log(Math.round(0.6)); //1

console.log(Math.round(0.5)); //1

console.log(Math.round(0.4)); //0

max()

Parametre olarak verilen sayıların en büyüğünü döndürür.

min()

Parametre olarak verilen sayıların en küçüğünü döndürür.

Example

console.log(Math.max(-10, 0, 10)); //10

console.log(Math.min(-10, 0, 10)); //-10

random()

0 ile 1 arasında rastgele sayı üretir.

Example

console.log(Math.random()); //0.3146060952880738

console.log(Math.random()); //0.7054790709570331

console.log(Math.random()); //0.8383301731900936

random() fonksiyonu farklı amaçlar için kullanılabilmektedir. Örneğin 0 ile 10 arasında(0 ve 10 dahil) bir tamsayı üretmek için random fonksiyonu 10 ile çarpılıp daha sonra yuvarlama fonksiyonlarından biri ile sayının ondalık kısmı atılabilir.

Example

console.log(Math.random() \* 10); //3.612652341030975

console.log(Math.round(Math.random() \* 10)); //7

Benzer şekilde 0 ile 100 arasında bir tamsayı üretmek için random fonksiyonu 100 ile, 0 ile 1000 arasında bir tamsayı üretmek için random fonksiyonu 1000 ile çarpılabilir.

Date

prototype

constructor

getDate()

Günü verir. (Dönebilecek değerler : 1-31)

getMonth()

Ayı verir. (Dönebilecek değerler : 0-11)

getFullYear()

Yılı verir.

getDay()

Haftanın kaçıncı günü olduğunu döndürür. (Dönebilecek değerler : 0-6, ilk gün Pazar)

Example

//konsola yazılan değerler sistem tarihine bağlıdır.(1 OCAK 2023)

let date = new Date();

console.log(date.getDate()); //1

console.log(date.getMonth()); //0

console.log(date.getFullYear()); //2023

console.log(date.getDay()); //0 -> Pazar

Example

/\*Aşağıdaki örnekte tarih bilgileri Date objesine parametre olarak verilen tarihten alınmıştır.(16 MAYIS 2008)\*/

let date = new Date("5/16/2008 23:15:00");

console.log(date.getDate()); //16

console.log(date.getMonth()); //4

console.log(date.getFullYear()); //2008

console.log(date.getDay()); //5

setDate()

Günü değiştirmek için kullanılır. (Beklenen değerler : 1-31)

setMonth()

Ayı değiştirmek için kullanılır. (Beklenen değerler : 0-11)

setFullYear(year, month, day)

Yılı değiştirmek için kullanılır. Üç parametreli kullanarak ay ve günü de değiştirebilirsiniz. İlk parametre zorunlu ikinci ve üçüncü parametreler opsiyoneldir.

Example

let date = new Date();

console.log(date.getDate(), date.getMonth(), date.getFullYear()); //1 0 2023

date.setDate(2);

date.setMonth(1);

date.setFullYear(1999);

console.log(date.getDate(), date.getMonth(), date.getFullYear()); //2, 1, 1999

getHours()

Saati verir. (Dönebilecek değerler : 0-23)

getMinutes()

Dakikayı verir. (Dönebilecek değerler : 0-59)

getSeconds()

Saniyeyi verir. (Dönebilecek değerler : 0-59)

getMilliseconds()

Milisaniyeyi verir. (Dönebilecek değerler : 0-999)

Example

//konsola yazılan değerler sistem saatine bağlıdır. (10:49:53.770)

let date = new Date();

console.log(date.getHours()); //10

console.log(date.getMinutes()); //49

console.log(date.getSeconds()); //53

console.log(date.getMilliseconds()); //770

Example

/\*Aşağıdaki örnekte saat, dakika, saniye ve milisaniye bilgileri Date objesine parametre olarak verilen tarihten alınmaktadır. (23:15:00.000) \*/

let date = new Date("5/16/2008 2008 23:15:00");

console.log(date.getHours()); //23

console.log(date.getMinutes()); //15

console.log(date.getSeconds()); //0

console.log(date.getMilliseconds()); //0

setHours()

Saati değiştirmek için kullanılır.

setMinutes()

Dakikayı değiştirmek için kullanılır.

setSeconds()

Saniyeyi değiştirmek için kullanılır.

setMilliseconds()

Milisaniyeyi değiştirmek için kullanılır.

Example

let date = new Date("5/16/2008 23:15:00");

console.log(date.getHours(), date.getMinutes(), date.getSeconds(), date.getMilliseconds()); //23, 15, 0, 0

date.setHours(22);

date.setMinutes(5);

date.setSeconds(31);

date.setMilliseconds(32);

console.log(date.getHours(), date.getMinutes(), date.getSeconds(), date.getMilliseconds()); //22, 5, 31, 32

toString()

ECMAScript 6

Arrow Functions

Example

let message = () => {

  console.log("Hello world!");

}

Example

let cube = (number) => {

  return (number \* number \* number);

}

* Tek parametreli fonksiyonlar için parantez zorunlu değildir.

Example

let cube = number => {

  return (number \* number \* number);

}

* Fonksiyon içerisinde tek bir işlem yapılıyorsa süslü parantezlere de gerek yoktur.

Example

let cube = number => (number \* number \* number);

Template Literal

Example

let product = {

  name: "Sweater",

  size: "L",

  price: 129.90

}

let productInfo = "Product Name : " + product.name + " Size : " + product.size + " Price : " + product.price;

console.log(productInfo);

Standart sözdizimle yazılmış yukarıdaki kodları template literal ile daha kolay ve düzenli bir şekilde oluşturabiliriz.

Example

let product = {

  name: "Sweater",

  size: "L",

  price: 129.90

}

let productInfo = `Product Name : ${product.name} Size : ${product.size} Price : ${product.price}`;

console.log(productInfo);

* Template literal içerisinde Javascript kodları kullanılabilmektedir.

Spread Operator

Example

let languages = ["C#", "Java", "Python", "C++"];

/\*languages[0], languages[1], languages[2], languages[3] şeklinde dizinin her elemanı virgül ile ayrılarak okunmaktadır\*/

console.log(...languages); //C# Java Python C++

OOP(Object Oriented Programming using ES6)

Example

class Employee{

  //constructor içinde tanımlanan özellik ve metodlar obje içinde oluşturulur.

  constructor(name, age, salary){

    //private field. Sadece constructor içinde kullanılabilir.

    let \_company = "Microsoft";

    //public fields

    this.name = name;

    this.age = age;

    this.salary = salary;

  }

  //Bu bölgede tanımlanan metodlar prototip içerisinde oluşturulur.

  showInfos(){

    console.log(this.name, this.age, this.salary);

  }

  //static değişken. obje üzerinden kullanılamaz. "Person.Count" şeklinde

//kullanabilirsiniz.

  static Count = 0;

  static UpdateCount(){

    this.Count++;

  }

}

Example

class Person{ //base class

  constructor(name, age){

    this.name = name;

    this.age = age;

  }

  showInfos(){

    console.log(this.name, this.age);

  }

}

class Employee extends Person{

  constructor(name, age, salary){

    super(name, age); //base class'ın constructor'ı çağırılıyor.

    this.salary = salary;

  }

  //showInfos metodu override ediliyor.

  showInfos(){

    console.log(this.name, this.age, this.salary);

    //base class'daki bir metodu kullanmak istediğimizde super anahtar kelimesinden yararlanıyoruz.

    super.showInfos();

  }

}

Callback Functions

Javascript kodları asenkron olarak çalışmaktadır. Bunu bir örnek üzerinden görebiliriz.

Example

function Process1(){

  setTimeout(function(){

    console.log("Process1");

  }, 2000)

}

function Process2(){

  console.log("Process2");

}

Process1();

Process2();

Önce Process2 daha sonra Process1 çalışmaktadır. Burada asenkron bir yapı söz konusudur.

Example

function Process1(callback){

  setTimeout(function(){

    console.log("Process1");

    callback();

  }, 2000)

}

function Process2(){

  console.log("Process2");

}

Process1(Process2);

Yukarıdaki örnekte callback fonksiyon mantığını kullanarak senkron bir yapı oluşturduk. Bu durumda önce Process1 daha sonra Process2 çalışacaktır.

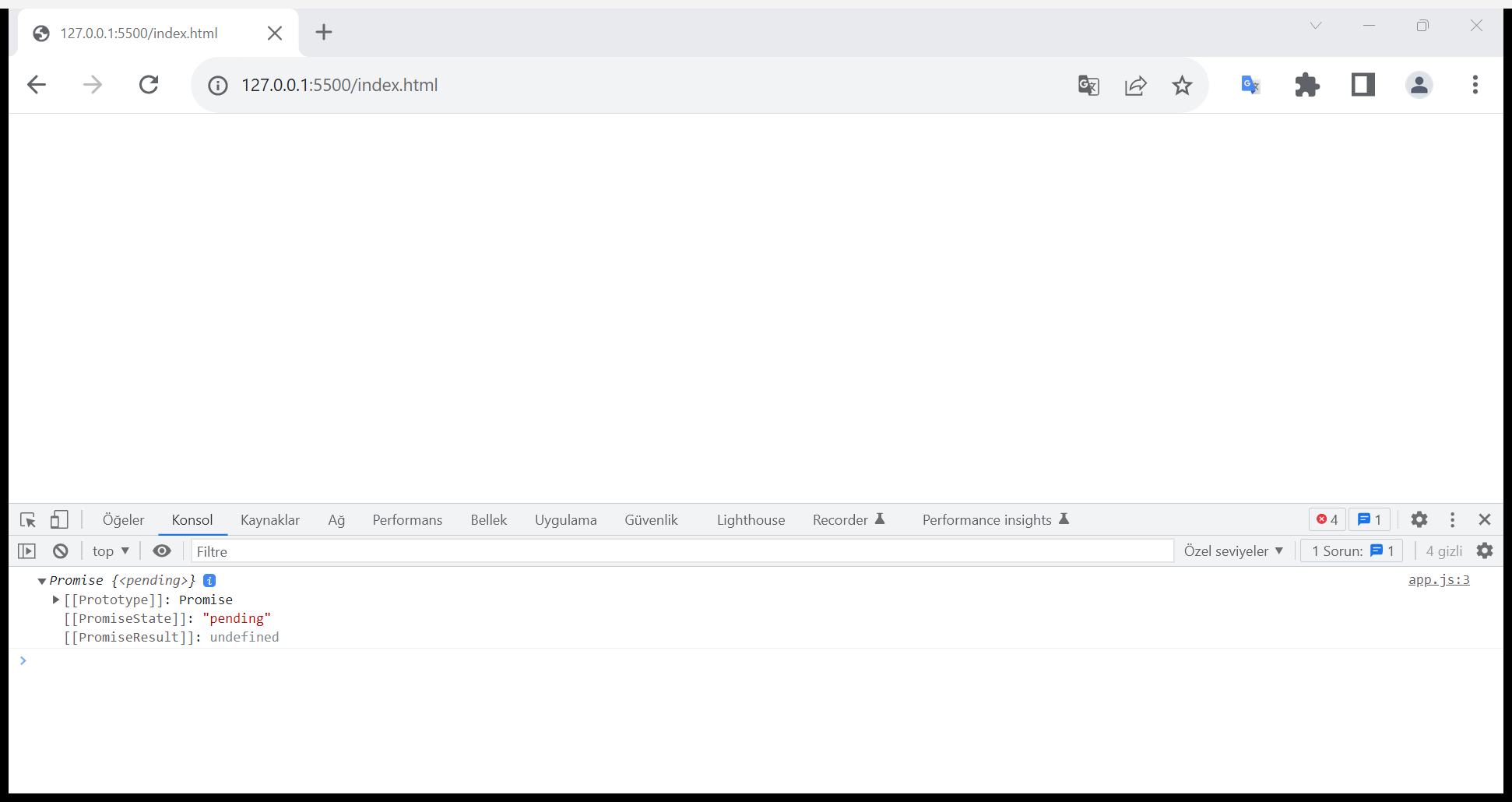
Promise Objesi – async, await

Promise objesi tanımlamak için Promise() constructor’ı kullanılmaktadır.

Example

let promise = new Promise((resolve, reject) => {});

console.log(promise);



Promise objesi içerisinde birtakım işlemler yapılır ve geriye olumlu ya da olumsuz bir sonuç dönülür. Olumlu sonuç dönmek için resolve, olumsuz sonuç dönmek için reject fonksiyonları kullanılmaktadır.

Promise objesinin state ve result adında iki özelliği bulunmaktadır. State özelliği başlangıçta pending, result özelliği ise undefined durumundadır. Bu sırada promise içerisinde zaman alan bazı işlemler yapılıyor olabilir. Örneğin :

Example

let promise = new Promise((resolve, reject) => {

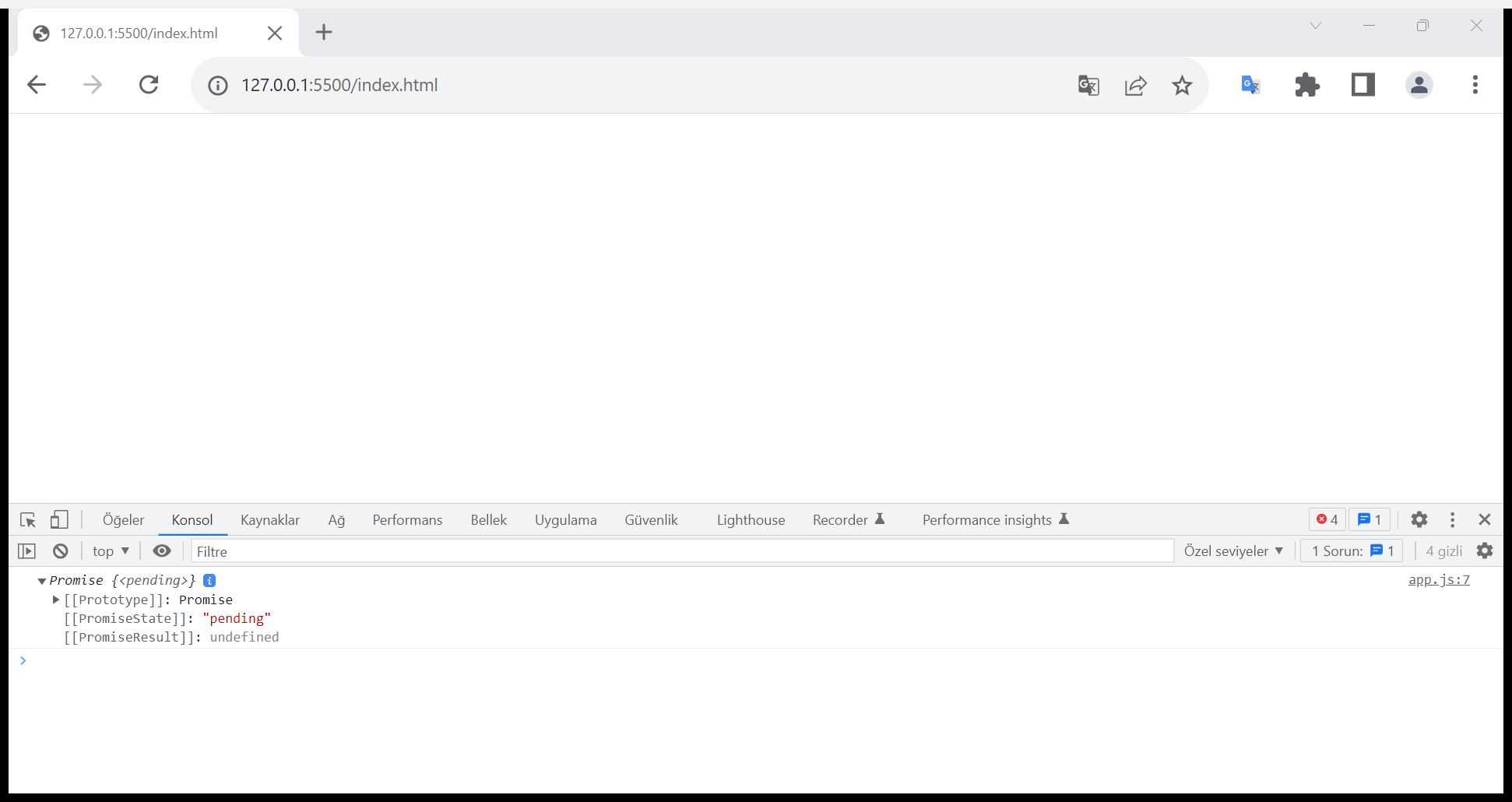
  setTimeout(() => {

    resolve("merhaba...") //Parametre kullanarak geriye değer döndürüyoruz.

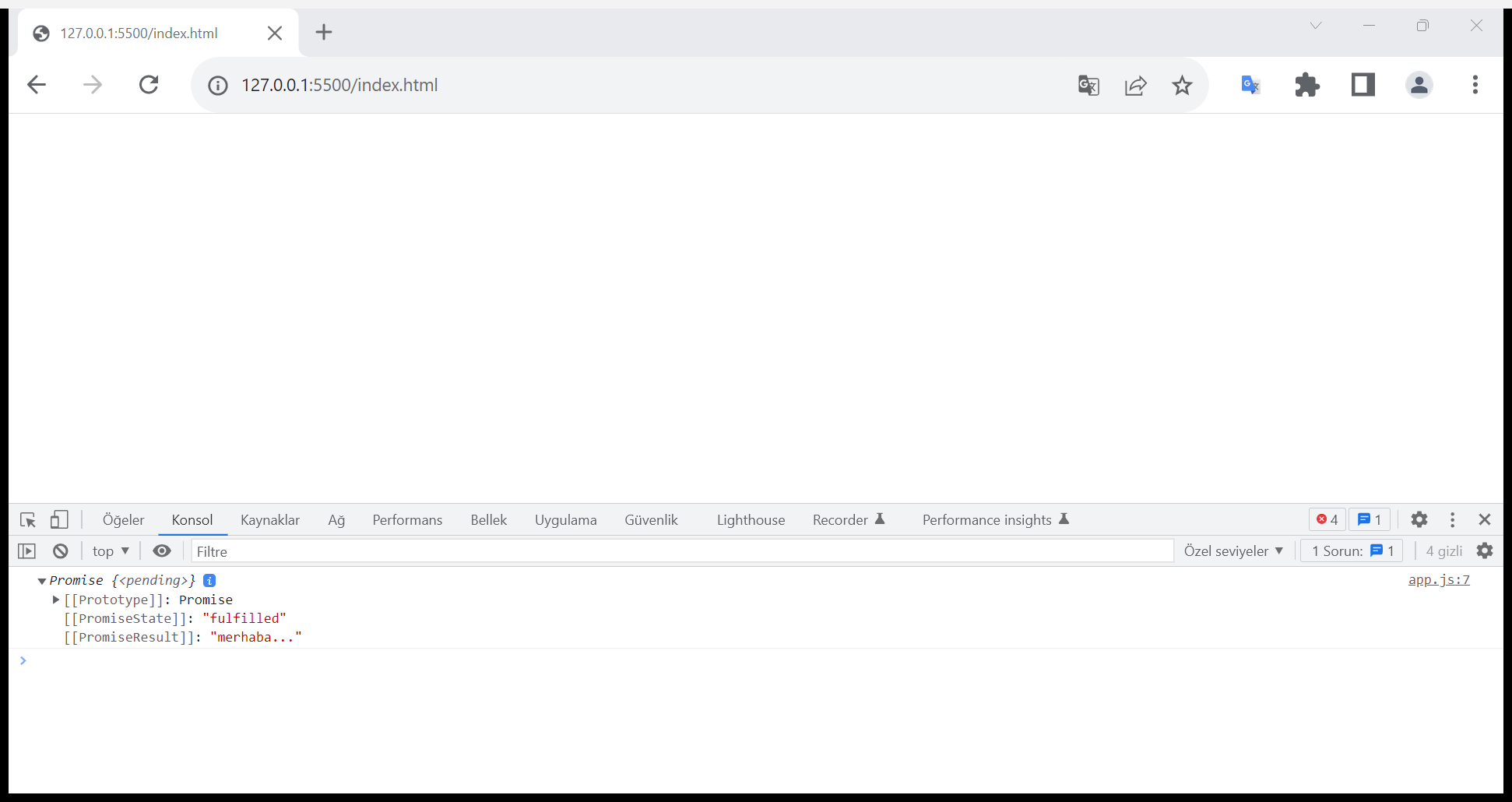
  }, 2000);

});

console.log(promise);



İki saniye sonra konsol çıktısına tekrar bakıyoruz.



state özelliği fulfilled, result özelliği ise geriye döndüğümüz değer olacaktır. Bu örnekte resolve fonksiyonunu kullandık ancak reject fonksiyonunu da kullanabilirdik. Bu durumda state özelliği rejected, result özelliği ise yine aynı şekilde geriye döndüğümüz değer olurdu.

Bir başka örnek ile bilgimizi pekiştirelim.

Example

function ageControl(age){

  return new Promise((resolve, reject) => {

  if(age >= 18) resolve(true);

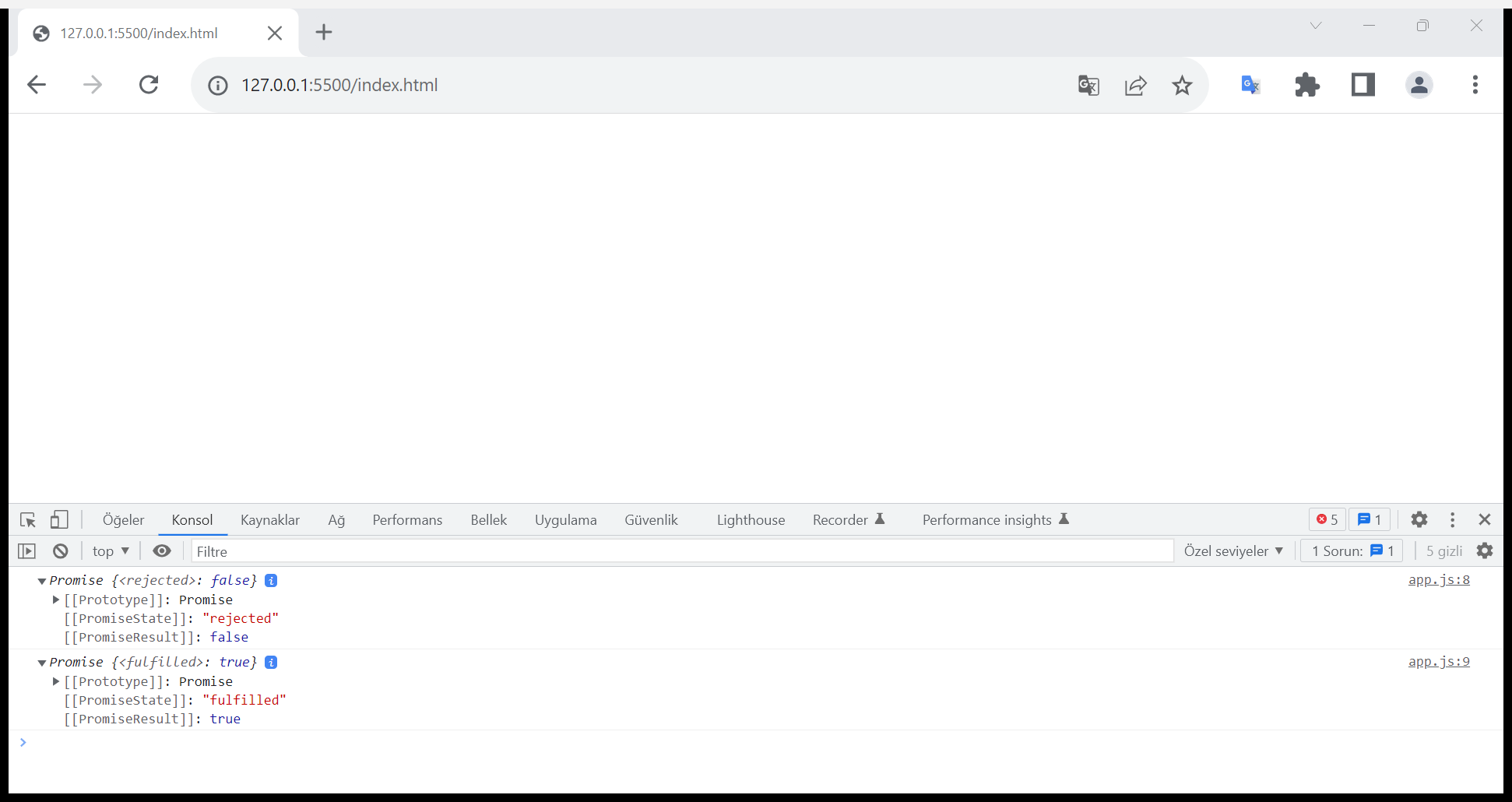
    reject(false);

  });

}

console.log(ageControl(17));

console.log(ageControl(18));



Promise objesinin result özelliğine doğrudan erişemiyoruz. Result’u okumak için then ve catch fonksiyonları kullanılmaktadır.

Example

function ageControl(age){

  return new Promise(function(resolve, reject){

  if(age >= 18) resolve(true);

  reject(false);

  });

}

ageControl(18)

.then((response) => {

  console.log(response);

})

.catch((err) => {

  console.log(err);

});

then içerisinden geriye tekrar değer gönderip bu değeri bir sonraki then içerisinden elde edebiliriz. Yani birden çok then kullanılabilir ancak catch bir kez kullanılabilmektedir.

Example

function ageControl(age){

  return new Promise(function(resolve, reject){

  if(age >= 18) resolve(true);

  reject(false);

  });

}

ageControl(18)

.then((response) => {

  console.log(response); //true

  return "Yetişkin";

})

.then((response) => {

  console.log(response); //Yetişkin

//Geriye Promise objesi de dönebiliyoruz.

  return new Promise((resolve, reject) => {

  reject("Error Message");

  });

})

.then((response) => {

  console.log(response); //reject döndüğümüz için burası çalışmaz.

})

.catch((err) => {

  console.log("Hata : " + err); //reject döndüğümüz için catch çalışır.

});

* Promise objesi callback fonksiyonlar yerine kullanılabilir.

Example

//Callback hell

function Process1(callback){

  setTimeout(function(){

    console.log("Process1 çalıştı");

    callback();

  },2000)

}

function Process2(callback){

  setTimeout(function(){

    console.log("Process2 çalıştı")

    callback();

  },2000)

}

function Process3(callback){

  setTimeout(function(){

    console.log("Process3 çalıştı");

    callback();

  },2000)

}

function Process4(){

  console.log("Process4 çalıştı")

}

Process1(function(){

  Process2(function(){

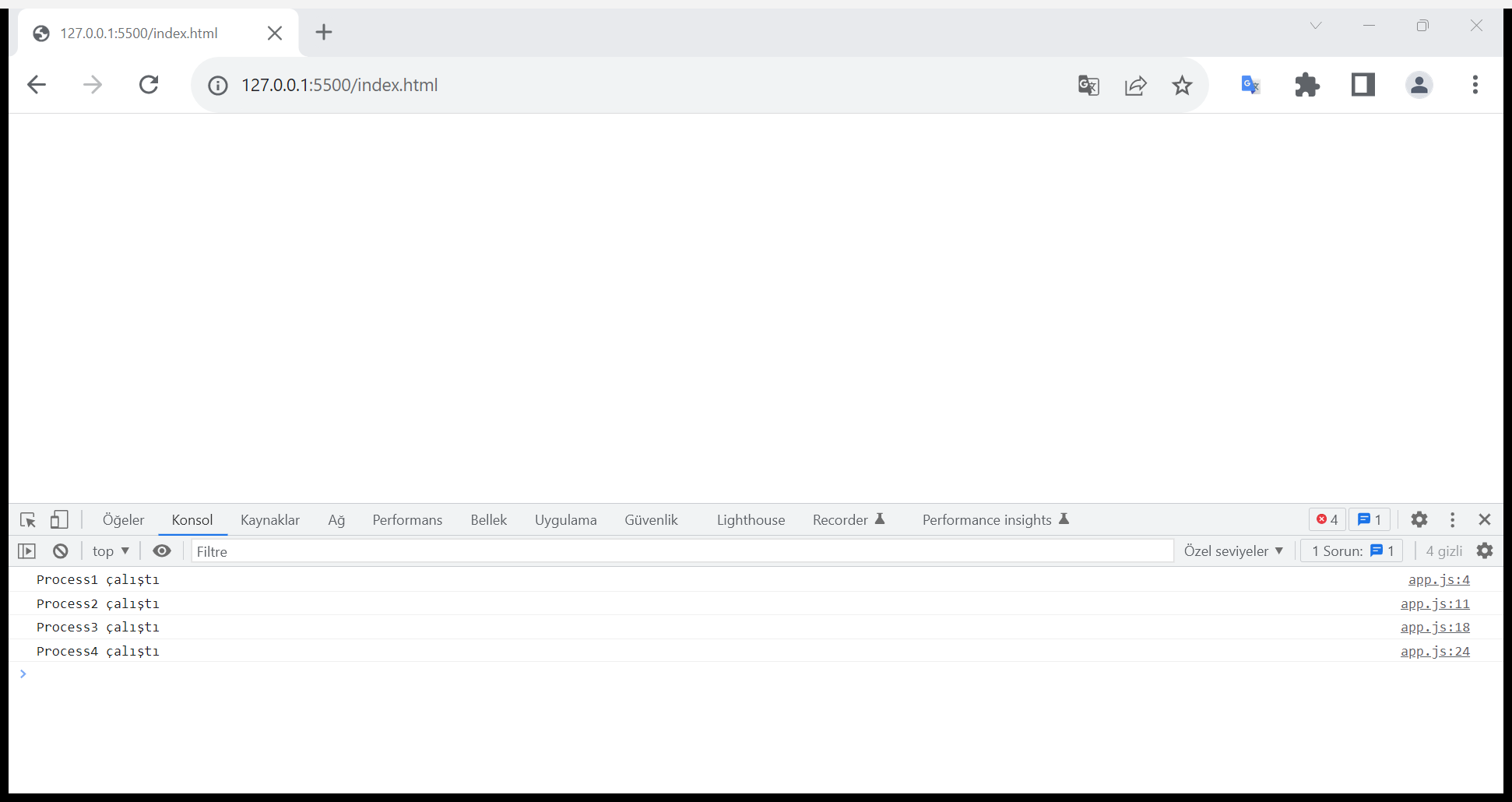
    Process3(function(){

      Process4();

    });

  });

});



Yukarıdaki işlemin aynısını promise’ler ile yapabiliriz.

Example

function Process1(){

  return new Promise((resolve, reject) => {

    setTimeout(function(){

      console.log("Process1 çalıştı");

      resolve();

    },2000)

  })

}

function Process2(){

  return new Promise((resolve, reject) => {

    setTimeout(function(){

      console.log("Process2 çalıştı");

      resolve();

    },2000)

  })

}

function Process3(){

  return new Promise((resolve, reject) => {

    setTimeout(function(){

      console.log("Process3 çalıştı");

      resolve();

    },2000)

  })

}

function Process4(){

  console.log("Process4 çalıştı");

}

Process1().then(Process2).then(Process3).then(Process4);

Aynı örnek üzerinde async ve await anahtar kelimelerini kullanabiliriz.

Example

function Process1(){

  return new Promise((resolve, reject) => {

    setTimeout(function(){

      console.log("Process1 çalıştı");

      resolve();

    },2000)

  })

}

function Process2(){

  return new Promise((resolve, reject) => {

    setTimeout(function(){

      console.log("Process2 çalıştı");

      resolve();

    },2000)

  })

}

function Process3(){

  return new Promise((resolve, reject) => {

    setTimeout(function(){

      console.log("Process3 çalıştı");

      resolve();

    },2000)

  })

}

function Process4(){

  console.log("Process4 çalıştı");

}

async function Run(){

  await Process1();

  await Process2();

  await Process3();

  Process4();

  //Geriye bir Promise objesi dönülmesi gerekmekte.

  return 1;

}

Run();

async, bir fonksiyonun geriye bir promise objesi döneceğini garantiler. await ise bir fonksiyonun geriye promise objesi döndürmesini bekler. await anahtar kelimesi sadece async fonksiyonlar içerisinde kullanılabilmektedir.

|  |
| --- |
| **NOT**  Promise objesi ECMAScript 6 ile birlikte gelmiştir.[Edge +]  async ve await anahtar kelimeleri ECMAScript 2017 ile birlikte gelmiştir.[Edge 15+] |

AJAX

AJAX(Asynchronous Javascript And XML - Asenkron Javascript ve XML) sayfa yenilenmeden asenkron olarak server ya da bir web api ile veri alışverişi yapmamızı sağlamaktadır.

* Veri alışverişi **XML** ya da **text** formatında yapılabilmektedir.
* AJAX işlemleri(GET, POST, PUT, DELETE request) için XMLHttpRequest objesi kullanılmaktadır.

Bu objenin özellik ve metodlarını konsol ekranında görüntüleyebiliriz.

Example

let xhr = new XMLHttpRequest();

console.log(xhr);

Örnek bir AJAX uygulaması aşağıda verilmiştir.

Example

let xhr = new XMLHttpRequest();

xhr.open("GET", "example.txt", true);

xhr.onreadystatechange = function(){

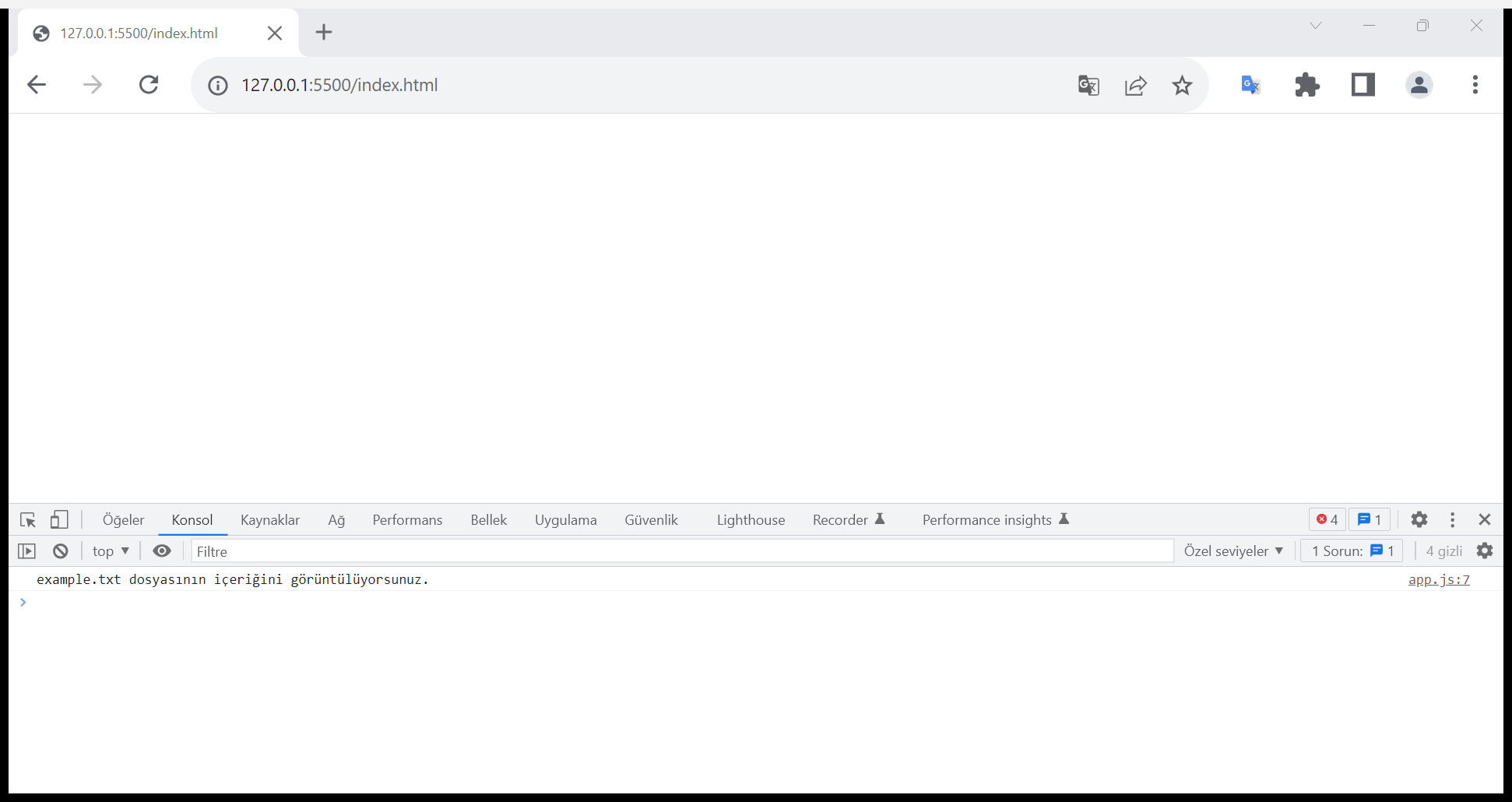
  if(this.readyState == 4 && this.status == 200){

    console.log(this.responseText);

  }

}

xhr.send();



Öncelikle bir XMLHttpRequest objesi oluşturuyoruz.

open(method, url, isAsync) metodu ile istekle ilgili birtakım bilgileri ayarlıyoruz.

İlk parametre isteğin türü. Veri çekmek için **GET**, veri göndermek için **POST**, güncelleme işlemleri için **PUT**, silme işlemi için **DELETE** kullanıyoruz.

İkinci parametre endpoint(URL).

Üçüncü parametre isteğin asenkron olup olmayacağı. Varsayılan değeri true olduğu için kullanılmasına gerek yoktur.

onreadystatechange, readyState değeri her değiştiğinde gerçekleşen olaydır.

readyState değerlerinin anlamları :

0 – open metodu kullanılmadan önce

1 – open metodu kullanıldıktan sonra

2 – istek alındı

3 – istek işleniyor

4 – istek tamamlandı ve response hazır

Sonuç olarak readystate değeri 4 olduğunda response’un hazır olduğunu anlıyoruz. Ancak bu aşamada status özelliğinin de kontrol edilmesi gerekmektedir. Çünkü sunucu response olarak bir hata mesajı döndürmüş olabilir.

status kodlarının anlamları :

202 – OK

404 – Not Found

|  |
| --- |
| **NOT**  http durum kodları ve anlamları :  <https://restfulapi.net/http-status-codes/>  <https://www.w3schools.com/tags/ref_httpmessages.asp> |

send() metodu ile isteği gönderiyoruz. (Sunucu tarafına veri gönderilirken bu metod send(string) şeklinde kullanılmaktadır)

* readyState ve status mantığını anlamak için aşağıdaki örneği inceleyebilirsiniz.

Example

let xhr = new XMLHttpRequest();

console.log(xhr.readyState); //0 -> open metodu kullanılmadan önce

xhr.open("GET", "https://jsonplaceholder.typicode.com/albums", true);

console.log(xhr.readyState); //1 -> open metodu kullanıldıktan sonra

xhr.onreadystatechange = function(){

    console.log(this.readyState); //2 //3 //4

    console.log(this.status); //200 //200 //200

}

xhr.send();

* XMLHttpRequest objesinin onprogress ve onload olayları bulunmaktadır. readyState değeri 3 olduğunda onprogress, 4 olduğunda ise onload olayı gerçekleşmektedir.

Example

let xhr = new XMLHttpRequest();

xhr.open("GET", "https://jsonplaceholder.typicode.com/albums", true);

xhr.onprogress = function(){

  console.log(this.readyState); //3

}

xhr.onload = function(){

  console.log(this.readyState); //4

}

xhr.send();

* responseText string türündeki verileri okumak için kullanılmaktadır.
* responseXML xml türündeki verileri okumak için kullanılmaktadır.
* abort(), isteği iptal eder. send() metodundan sonra kullanılmalıdır, önce kullanılırsa bir etkisi olmaz.
* getAllResponseHeaders() başlık bilgisinin tamamını döndürür.
* getResponseHeader(string) ilgili başlık bilgisini döndürür.
* setRequestHeader(header, value) gönderilecek verilere başlık bilgisi eklemek için kullanılır.

Example

let xhr = new XMLHttpRequest();

xhr.open("GET", "https://jsonplaceholder.typicode.com/albums", true);

xhr.onload = function(){

  if(this.status == 200){

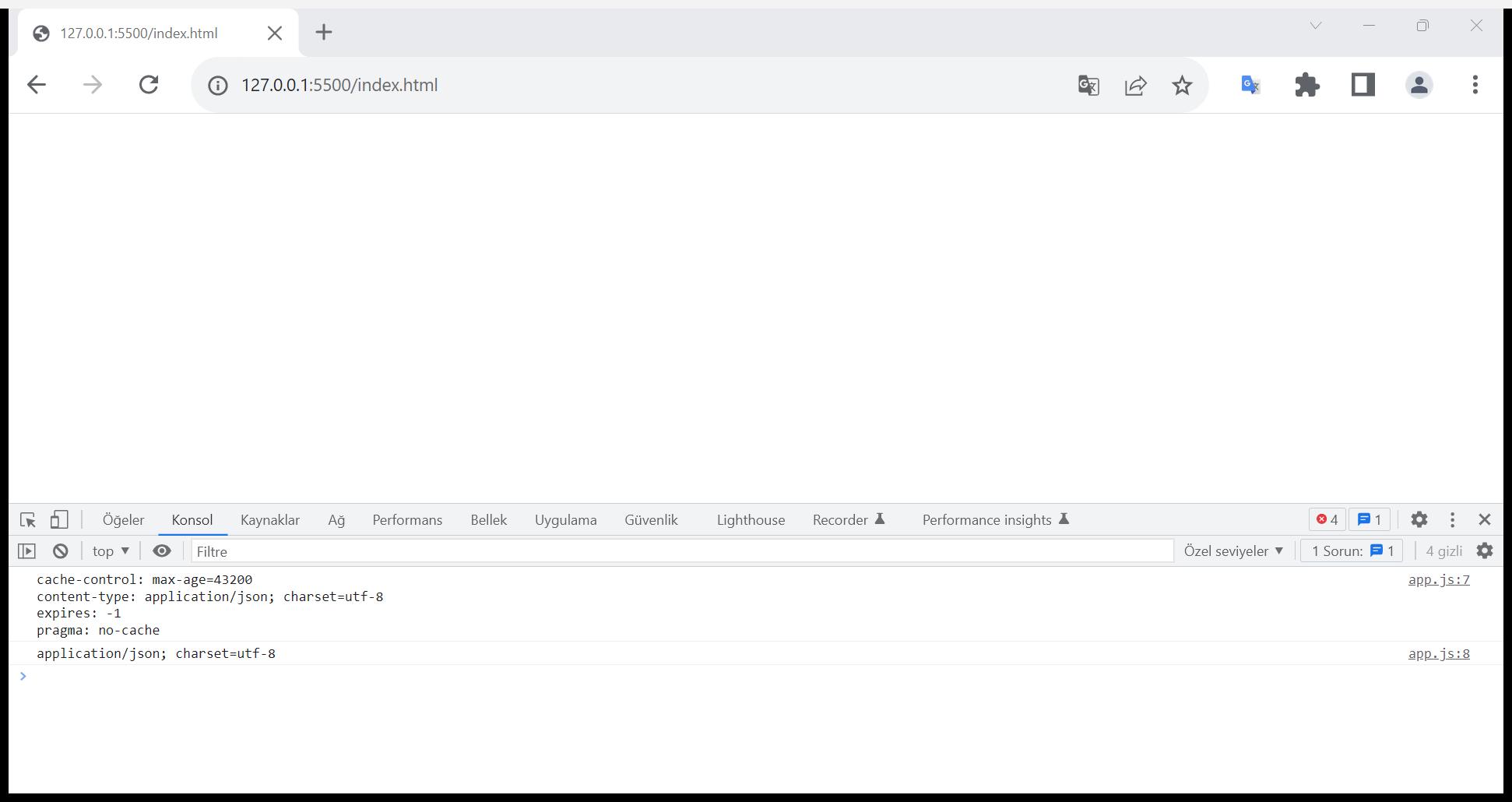
    console.log(this.getAllResponseHeaders());

    console.log(this.getResponseHeader("content-type"));

  }

}

xhr.send();



Gelişmiş bir AJAX uygulaması ile konuyu bitiriyoruz.

Example

class AJAX{

  constructor(){

      this.xhr = new XMLHttpRequest();

  }

  get(url, callback){

      this.xhr.open("GET", url, true);

      this.xhr.onload = function(){

          if(this.status == 200){

              callback(this.responseText, null);

          }

          else{

              callback(null, "Bir hata meydana geldi : " + this.status);

          }

      }

      this.xhr.send();

  }

  post(url, data, callback){

      this.xhr.open("POST", url, true);

      this.xhr.onload = function(){

          if(this.status == 201){

              callback(this.responseText, null);

          }

          else{

              callback(null, "Bir hata meydana geldi : " + this.status);

          }

      }

      this.xhr.setRequestHeader("Content-type", "application/json; charset=UTF-8");

      this.xhr.send(data);

  }

  put(url, data, callback){

      this.xhr.open("PUT", url, true);

      this.xhr.onload = function(){

          if(this.status == 200){

              callback(this.responseText, null);

          }

          else{

              callback(null, "Bir hata meydana geldi : " + this.status);

          }

      }

      this.xhr.setRequestHeader("Content-type", "application/json; charset=UTF-8");

      this.xhr.send(data);

  }

  delete(url, callback){

      this.xhr.open("DELETE", url, true);

      this.xhr.onload = function(){

          if(this.status == 200){

              callback(this.responseText, null);

          }

          else{

              callback(null, "Bir hata meydana geldi : " + this.status);

          }

      }

      this.xhr.send();

  }

}

var ajax = new AJAX();

// ajax.get("https://jsonplaceholder.typicode.com/albums", function(response, error){

//     console.log(response);

//     console.log(error);

// });

// ajax.get("https://jsonplaceholder.typicode.com/albums/1", function(response, error){

//     console.log(response);

//     console.log(error);

// });

// ajax.post("https://jsonplaceholder.typicode.com/albums",

//     JSON.stringify({title: 'Resim', userId: 10}),

//     function(response, error){

//         console.log(response);

//         console.log(error);

// });

// ajax.put("https://jsonplaceholder.typicode.com/albums/1",

//     JSON.stringify({title: 'Resim', userId: 10}),

//     function(response, error){

//         console.log(response);

//         console.log(error);

// });

// ajax.delete("https://jsonplaceholder.typicode.com/albums/1", function(response, error){

//     console.log(response); //{} ->Geriye boş obje döner.

//     console.log(error);

// });

JSON

Javascript Object Notation(Javascript Obje Notasyonu). JSON, javascript obje sözdizimi ile yazılmış bir metindir.

* Veri saklamak ve taşımak için kullanılmaktadır.
* JSON dosyaları **.json** uzantılıdır.
* JSON metinlerinin MIME türü **application/json**’dır.
* AJAX ile sadece string ve xml türünde veriler taşınabilmektedir. JSON da bir string olduğu için JSON formatındaki response okunurken responseText kullanılmaktadır.
* Storage’a sadece string türünde veriler kaydedilebilmektedir. Obje ve diziler storage’a JSON formatında eklenebilmektedir. Bunun için JSON.stringify() kullanın.
* JSON’a alternatif olarak XML kullanabilirsiniz. Ancak JSON’ın XML’e karşı birçok üstünlüğü bulunmaktadır. Yine de ikisi arasındaki karşılaştırmayı görmek isteyebilirsiniz :

<https://www.w3schools.com/js/js_json_xml.asp>

Aşağıda JSON formatında bir string görüyorsunuz.

Example

let json = '{"name": "John", "age": 30, "city": "New York"}';

console.log(typeof json); //string

JSON Sözdizim Kuralları

* JSON’da isim değer çiftleri string türünde olmalıdır.
* JSON’da değerler şu veri türlerinden biri olmalıdır :

string

“name”: “John”

number

“age”: 30

boolean

“sale”: true

array

“employees”: [“John”, “Anna”, “Peter”]

object

“employee”: {“name”: “John”, “age”: 30, “city”: “New York”}

null

“job”: null

* JSON’da değerler şu veri türlerinden biri olamaz : function, Date(), undefined

JSON.parse()

JSON verisini ayrıştırmak için kullanılır. JSON ayrıştırıldığında bir javascript objesi haline gelir.

Example

var json = '{"name": "John", "age": 30, "city": "New York"}';

var obj = JSON.parse(json);

console.log(typeof json); //string

console.log(typeof obj); //object

console.log(json); //{"name": "John", "age": 30, "city": "New York"}

console.log(obj); //{name: 'John', age: 30, city: 'New York'}

JSON.stringify()

Javascript objelerini(ya da dizilerini) JSON formatında string’e dönüştürür.

Example

var obj = {name: "John", age: 30, city: "New York" };

var json = JSON.stringify(obj);

console.log(obj); //{name: 'John', age: 30, city: 'New York'}

console.log(json); //{"name":"John","age":30,"city":"New York"}

Example

var employees = ["John", "Peter", "Sally", "Jane"];

console.log(JSON.stringify(employees)); //["John","Peter","Sally","Jane"]

* JSON verisi içerisinde tarih objelerine izin verilmediği için JSON.stringify() tarih objelerini string’e dönüştürür. string datayı alıcıda tekrar bir tarih objesine dönüştürebilirsiniz.

Example

var obj = {name: "John", today: new Date(), city: "New York"};

var json = JSON.stringify(obj);

console.log(json); //{"name":"John","today":"2023-01-01T15:52:05.164Z","city":"New York"}

* JSON verisi içerisinde fonksiyonlara izin verilmediği için JSON.stringify() fonksiyonları siler.

Example

var obj = {name: "John", city: "New York", foo: function(){}};

var json = JSON.stringify(obj);

console.log(json); //{"name":"John","city":"New York"}

Fonksiyonları string’e çevirip daha sonra JSON.stringify() kullanın. Alıcıda eval() ile yeniden fonksiyona dönüştürebilirsiniz.

Example

var obj = {name: "John", city: "New York", foo: function(){}};

obj.foo = obj.foo.toString();

var json = JSON.stringify(obj);

console.log(json); //{"name":"John","city":"New York","foo":"function(){}"}

XML

Extensible Markup Language(Genişletilebilir İşaretleme Dili).

* Veri saklamak ve taşımak için kullanılmaktadır.
* XML **W3C** tarafından geliştirilmiştir.
* XML dosyaları **.xml** uzantılıdır.
* XML dosyalarının MIME türü **text/xml** ya da **application/xml** olabilmektedir.

Aşağıda örnek bir XML dosyası görüyorsunuz.

Example

<?xml version="1.0" encoding="UTF-8" ?>

<employees>

  <employee>

    <name>Eren</name>

    <age>29</age>

  </employee>

  <employee>

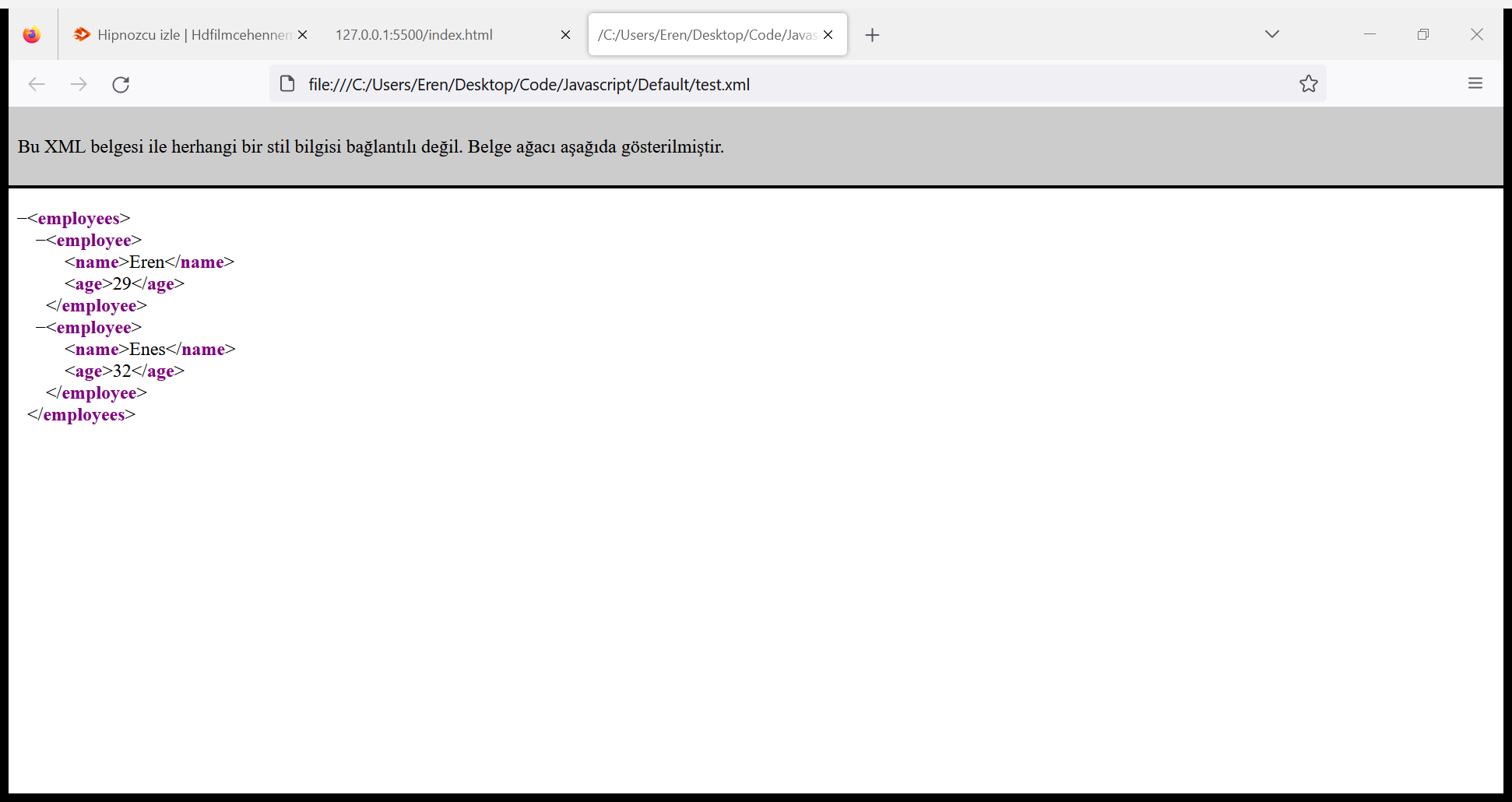
    <name>Enes</name>

    <age>32</age>

  </employee>

</employees>

Bu XML belgesini aşağıdaki gibi internet tarayıcısı üzerinde görüntüleyebilirsiniz.



* En dıştaki elemente **root element**, root element içindeki elementlere de **child element** denmektedir.
* Javascript ile **XML DOM** kullanarak XML verileri üzerinde dolaşabilirsiniz.

Example

function getXMLData(callback){

let xhr = new XMLHttpRequest();

xhr.open("GET", "./example.xml", true);

xhr.onload = function(){

if(this.status == 200){

callback(this.responseXML, null);

}

else{

callback(null, "Bir hata meydana geldi : " + this.status);

}

}

xhr.send();

}

getXMLData(function(response, error){

let result = response; //XML DOM object

let elements = result.childNodes[0].children;

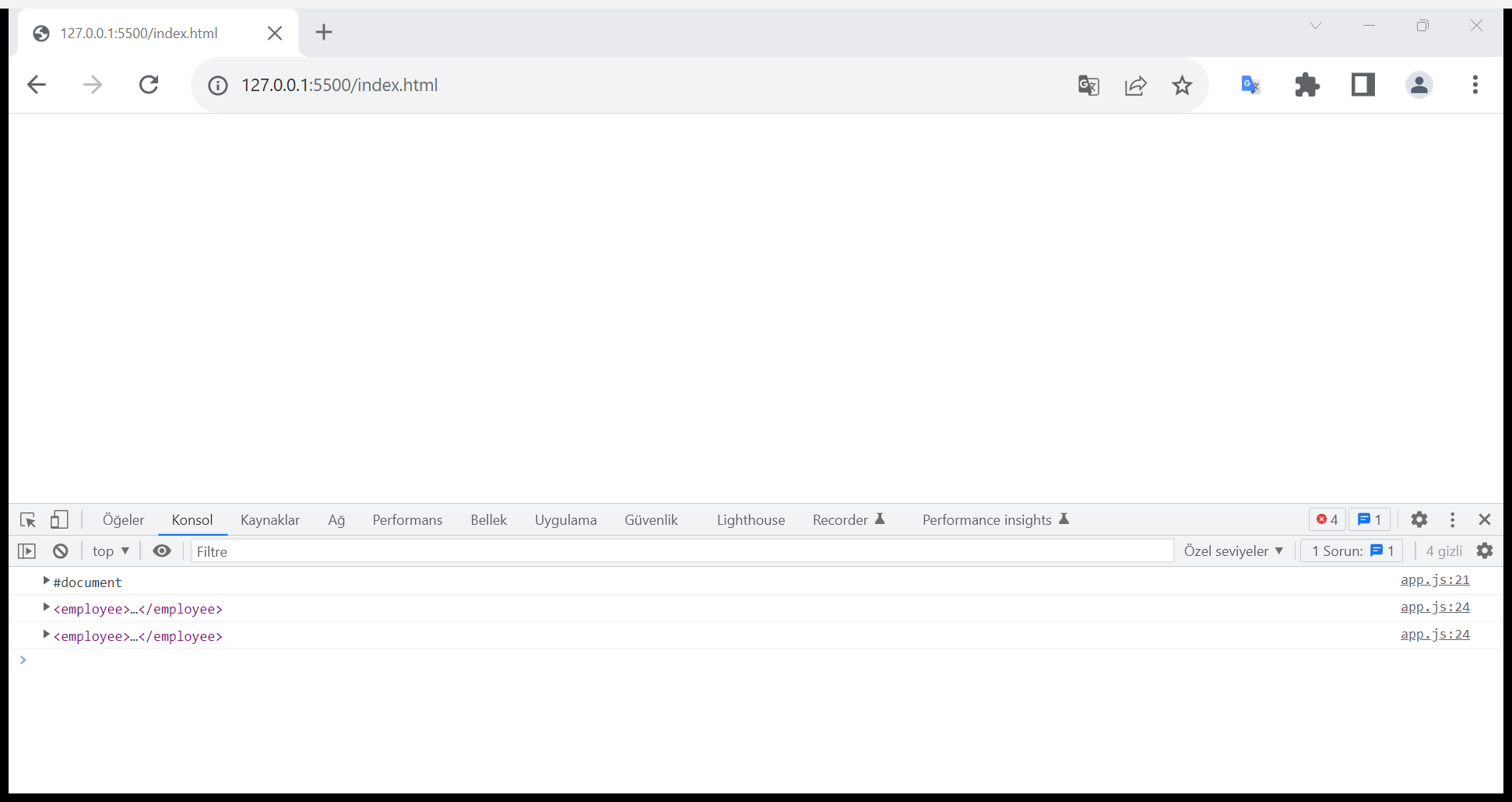
console.log(result);

for(var i = 0; i < elements.length; i++){

console.log(elements[i]);

}

});



* Javascript ile string formatında bir XML veriyi XML DOM objesine yükleyerek işlemlerinizi gerçekleştirebilirsiniz.

Example

var data = "<employees><employee><name>Eren</name><age>29</age></employee></employees>";

var parser = new DOMParser();

var result = parser.parseFromString(data, "text/xml"); //XML DOM object

console.log(result);

XML Sözdizim Kuralları

* XML dosyaları mutlaka root elemente sahip olmalıdır.
* Boş element oluşturabilirsiniz.
* Açılan her element kapatılmalıdır.
* Elementlerin kapanış sırası önemlidir.
* Element isimleri büyük küçük harfe duyarlıdır.
* Attribute değerleri tırnak içinde yazılmalıdır. Tek ya da çift tırnak kullanabilirsiniz.
* XML elementleri isimlendirilirken harf, rakam veya özel karakterler(- ve \_) kullanılabilir. Element isimleri rakam ile başlayamaz.
* XML dosyaları içinde açıklama metinlerine yer vermek için <!-- --> kullanılır.
* XML verisi içerisinde bazı özel karakterlerin kullanımı hataya neden olur. Bu özel karakterleri doğrudan kullanmak yerine entity referanslarını kullanabilirsiniz.

< : &lt;

> : &gt;

& : &amp;

“ : &quot;

‘ : &apos;