**İçerik(Index)**

Javascript:

1..........................[Javascript Nedir?](#JavascriptNedir)

2.....................[Kod Yazma Notasyonları](#KodYazmaNotasyonları)

3.....................[İsimlendirme Kuralları](#İsimlendirmeKuralları)

4...................[Veri Tipleri(Data Types)](#VeriTipleri)

5.....................[Değişkenler(Variables)](#Değişkenler)

6...................[Yararlı Konsol Komutları](#YararlıKonsolKomutları)

7............................[Tip Dönüşümleri](#TipDönüşümleri)

8......[Kaçış Karakterleri(Escape Characters)](#KaçışKarakterleri)

9.............[Yorum Satırları(Comment Lines)](#YorumSatırları)

10....................[Operatörler(Operators)](#Operatörler)

11....[Karar Yapıları(Conditional Statements)](#KararYapıları)

12...........................[Döngüler(Loops)](#Döngüler)

13..................................[Deyimler](#Deyimler)

14............................[İşlem Önceliği](#İşlemÖnceliği)

15...................[Fonksiyonlar(Functions)](#Fonksiyonlar)

16..[Global Fonksiyonlar ve Global Özellikler](#GlobalFonksiyonlarGlobalOzellikler)

17...........................[Diziler(Arrays)](#Diziler)

18.............................[Hata Yönetimi](#HataYönetimi)

19.............................[ECMAScript 6+](#ECMAScript6)

20.....................[Callback Fonksiyonlar](#CallbackFonksiyonlar)

21..........[OOP(Object Oriented Programming)](#OOP)

22.......................[Javascript Objeleri](#JavascriptObjeleri)

23.............[Promise Objesi – async, await](#PromiseObjesiAsyncAwait)

24......................[Asenkron Programlama](#AsenkronProgramlama)

25......................................[JSON](#JSON)

26.......................................[XML](#XML)

**1.Javascript Nedir?**

Javascript browser tabanlı bir programlama dilidir.

**‘**Dinamik(animasyonlu, kullanıcı ile etkileşimli) web sayfaları oluşturmak için kullanılır.

**‘**Browser tabanlı bir dil olduğu için Javascript ile **doğrudan** desktop, mobile ve server tabanlı uygulamalar geliştirilemez. Doğrudan dedim çünkü Node.js, React, Electron gibi bazı Javascript kütüphaneleri kullanarak bu tür uygulamalar üretebiliyoruz.

**‘**Brendan Eich tarafından geliştirilmiştir.

**‘**İlk sürümü 1995 yılında yayınlanmıştır.

**‘**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 doğru çalıştırılır.

**‘**Javascript 1997 yılında ECMA standardı olmuştur ve şuan resmi adı **ECMAScript**’tir. ECMA, yani European Computer Manufacturers Association(Avrupa Bilgisayar Üreticileri Birliği) bilgisayar sektöründe üretilen ürünler için standartlar oluşturan bir kuruluştur.

**‘**Javascript’in günümüze kadar birçok sürümü yayınlandı ve bu sürümlerle dile yeni yetenekler kazandırıldı. Javascript sürümleri hakkında detaylı bilgi için <https://www.w3schools.com/js/js_versions.asp> adresini ziyaret edebilirsiniz.

**2.Kod Yazma Notasyonları**

Aşağıdaki notasyonları kullanarak kodlarınızın daha okunabilir olmasını sağlayabilirsiniz.

**‘**CamelCase : Kelimeler bitişik ve her kelimenin ilk harfi büyük yazılır.

class Person{}

İlk kelimenin ilk harfi küçük yazılabilir.

let firstName = "Julia";

**‘**Snake\_Case veya snake\_case : Kelimeler alt tire ile birbirine bağlanır.

**‘**SCREAMING\_SNAKE\_CASE : Bazı programlama dillerinde sabitler için kullanılır.

**‘**kebap-case : Tüm harfler küçük olarak yazılır ve kelimeler arasında tire bulunur. Javascript bu notasyonu desteklemez.

**3.İsimlendirme Kuralları**

Javascript’te değişken, fonksiyon ve class isimlendirmelerinde aşağıdaki kurallar geçerlidir.

**‘**İsimler rakam ile başlayamaz ancak rakam içerebilirler.

**‘**Özel karakterler kullanılamaz. (\_ , $ hariç)

**‘**Türkçe’ye özel karakterlerin kullanılması tavsiye edilmez.

**‘**Anahtar kelimeler(var, function, class vb.), deyimler(continue, break, return) ve rezerve edilmiş kelimeler de isimlendirmelerde kullanılamazlar.

Rezerve edilmiş kelimeler : <https://www.w3schools.com/js/js_reserved.asp>

**4.Veri Tipleri(Data Types)**

Programlama dillerinde veriler string, number, boolean gibi belirli adlarla isimlendirilmişlerdir. Bu adlar veri tipi olarak ifade edilmektedir. Örneğin “Elma” şeklindeki bir ifade string olarak adlandırılmış ve bu noktada **Elma** bir veri **string** ise onun tipidir.

Javascript’te tipler ilkel tipler ve referans tipler olmak üzere ikiye ayrılır.

**Primitive Types(İlkel Tipler)**

string, number, boolean, null, undefined

string

Metinsel veriler string türündedir. Çift veya tek tırnak içerisinde yazılabilirler.

var firstName = "Eren";

var lastName = 'Satı';

console.log(typeof firstName); *//string*

number

Sayısal veriler number türündedir. Ayrıca Javascript’te özel anlamı olan NaN, Infinity ve -Infinity değerleri de number türündedir.

var age = 27;

const PI = 3.14; *//Virgüllü sayılar virgülle değil nokta ile yazılır.*

var hex = 0xFF; *//Hexadecimal sayıların önünde 0x bulunur.*

console.log(age, PI, hex); *//27 3.14 255*

console.log(typeof age); *//number*

console.log(typeof PI); *//number*

console.log(typeof hex); *//number*

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*

boolean

true ve false değerleri boolean türündedir.

var is\_equal = true;

console.log(typeof is\_equal); *//boolean*

undefined

undefined değeri undefined türündedir. Undefined’in Türkçesi **tanımsız** demektir. Değer ataması yapılmamış değişkenler undefined olur. Ayrıca değişkenlere undefined değeri verilerek içerikleri sıfırlanabilmektedir.

var firstName = undefined

var lastName;

console.log(firstName); *//undefined*

console.log(lastName); *//undefined*

console.log(typeof firstName); *//undefined*

console.log(typeof lastName); *//undefined*

null

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

var productName = null;

console.log(productName); *//null*

console.log(typeof productName); *//object*

**Referance Types(Referans Tipler)**

object, function

object

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

var obj = {};

var arr = [];

console.log(typeof obj); *//object*

console.log(typeof arr); *//object*

function

Fonksiyonlar function türündedir.

function addNumbers(){}

console.log(typeof addNumbers); *//function*

**İlkel ve Referans Tiplerin Farklı Davranışları**

**‘**İlkel tiplerde içerik kopyalanır, referans tiplerde ise bellek adresi aktarılır.

*//İlkel tip*

var x = 1;

var y = x;

x = 2;

console.log(y); *//1 ->Değer değişikliğinden y değişkeni etkilenmedi.*

*//Referans tip*

var a = [1, 2, 3];

var b = a;

a.push(4);

*//İçerik değişikliği b değişkenini de etkiler. Çünkü aynı bellek bölgesini gösteriyorlar*

console.log(b); *//1, 2, 3, 4*

*//NOT : = operatörü ile değer ataması yeniden yapılırsa j değişkeni bundan etkilenmez.*

*//i değişkeni farklı bir bellek bölgesinde yeniden tanımlanır.*

var i = [1, 2, 3];

var j = i;

i = [10, 20, 30];

console.log(j); *//1, 2, 3*

**‘**İlkel tiplerde içerik kontrol edilir, referans tiplerde ise adresler.

*//İlkel tip*

var x = 10;

var y = 10;

console.log(x===y); *//true*

*//Referans tip*

var a = [1, 2, 3];

var b = [1, 2, 3];

console.log(a==b); *//false*

console.log(a===b); *//false*

var i = [1, 2, 3];

var j = i;

console.log(i==j); *//true*

console.log(i===j); *//true*

**5.Değişkenler(Variables)**

Değişkenler bilgisayar belleğinde(RAM) veri saklamak için kullanılırlar.

**‘**Javascript’te değişkenler var, let, const anahtar kelimeleri ile tanımlanabilir.

**‘**Değer ataması yapılmamış değişkenler undefined olur.

**‘**Bir javascript değişkeni değer atanmaksızın yeniden tanımlanırsa değerini kaybetmez(undefined olmaz).

var number1;

number1 = 10;

var number1;

console.log(number1); *//10*

*//Bunun yerine undefined değeri vererek değişkenin içeriğini sıfırlayabilirsiniz*

var number2;

number2 = 20;

number2 = undefined;

console.log(number2); *//undefined*

**‘**const anahtar kelimesi ile oluşturulan değişkenler sabit(içeriği değiştirilemeyen) değişkenlerdir.

const name = "Eren";

const numbers = [1, 2, 3];

const person = {name: "Ahmet", age: 25}

const Product = function(ProductName){

  this.ProductName = ProductName;

}

*//Yanlışlar ve doğrular*

const number; *//Yanlış. İlk değer ataması zorunludur.*

name = "Melek"; *//Yanlış. Değer değişikliği yapılamaz.*

numbers = [10, 20, 30]; *//Yanlış. Sadece içerik değişebilir.*

numbers[0] = 10; *//Doğru. İlk elemanın değeri(1) 10 olarak değişir.*

numbers.push(4); *//Doğru. Dizinin sonuna 4 değeri eklenir.*

numbers.unshift(); *//Doğru. Dizinin ilk elemanını siler.*

person = {city: "İstanbul"} *//Yanlış. Sadece içerik değişebilir.*

person.city = "İstanbul"; *//Doğru.*

delete person.age; *//Doğru*

Product = function(){} *//Yanlış. Yeniden tanımlanamaz.*

Product.Count = 0; *//Doğru.*

var**,** let**,** const **Farkı**

**‘**let ve const ile tanımlanan değişkenler yeniden tanımlanamazlar.

var a = 1;

let b = 2;

const c = 3;

*//Değer değişikliği yapalım.*

a = 4; *//Doğru*

b = 5; *//Doğru*

c = 6; *//Yanlış*

*//Yeniden tanımlayalım.*

var a = 10; *//Doğru.*

let b = 20; *//Yanlış. Yeniden tanımlanamaz.*

const c = 30; *//Yanlış. Yeniden tanımlanamaz.*

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

*//Global scope*

var a = 1;

let b = 2;

const c = 3;

function test\_function(){ *//Function scope*

  var a = 4;

  let b = 5;

  const c = 6;

  console.log(a, b, c); *//4, 5, 6*

*//Global scope'daki a değişkeni var ile tanımlandığı için this anahtar kelimesiyle*

*//ulaşabilirsiniz.*

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

}

test\_function();

*//Fonksiyon içerisinde aynı isimde değişkenler tanımlamamıza rağmen global scope'daki*

*//değerler değişmedi. Çünkü Function scope'da tanımlanan değişkenler sadece ilgili*

*//fonksiyon içinde 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.

*//Global scope*

var a = 1;

let b = 2;

const c = 3;

*//Block Scope*

{

  var a = 4;

  let b = 5;

  const c = 6;

*//Block Scope tıpkı Function Scope gibi davrandı ancak bir farkla! Sonraki konsol*

*//çıktısına bakınız*

  console.log(a, b, c); *//4, 5, 6*

}

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

*//değiştirdi.*

console.log(a, b, c); *//4, 2, 3*

**‘**let ve const anahtar kelimeleri [**IE 11+]** tarafından desteklenmektedir.

**if-else**, **switch-case** ve **döngü blokları** da block scope kapsamındadır.

**6.Yararlı Konsol Komutları**

Aşağıdaki komutlar Javascript dosyaları içerisinden veya geliştirici konsolundan elle girilerek kullanılabilir.

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

console.log(this); *//window objesini verir.*

console.log(true); *//true*

console.log("Deneme yazısı..."); *//Deneme yazısı...*

console.log([1, 2, 3]); *//[1, 2, 3]*

**‘**console.clear() : Konsolu temizlemek için kullanılır.

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

console.error("Hata meydana geldi."); *//Hata meydana geldi.*

console.error(1); *//1*

console.error([1,2]); *//[1,2]*

console.error(this); *//Window{parent : Window, ...}*

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

console.warn("warning text"); *//warning text*

Konsol komutları hakkında daha fazla bilgi için <https://developer.mozilla.org/en-US/docs/Web/API/Console> adresine bakınız.

**7.Tip Dönüşümleri**

**‘**String() fonksiyonu : Parametre olarak verilen değeri string tipine dönüştürüp geri gönderir.

var number = String(25);

const PI = String(3.14);

var is\_equal = String(true);

var language = String("Javascript");

var empty = String(null);

console.log(number +"-"+ typeof number); *//25-string*

console.log(PI +"-"+ typeof PI); *//3.14-string*

console.log(is\_equal +"-"+ typeof is\_equal); *//true-string*

console.log(language +"-"+ typeof language); *//Javascript-string*

console.log(empty +"-"+ typeof empty); *//null-string*

var str = String([1, 2, 3, true, "Apple"]);

console.log(str +"=>"+ typeof str); *//1,2,3,true,Apple=>string*

var str = String(undefined);

console.log(str +"=>"+ typeof str); *//undefined=>string*

var str = String(function test\_function(){});

console.log(str +"=>"+ typeof str); *//function test\_function(){}=>string*

**‘**parseInt() fonksiyonu : Parametre olarak verilen değeri number tipine dönüştürüp geri gönderir. Tamsayılar için kullanılır. Virgüllü sayılarda virgülden sonrası atılır.

console.log(typeof parseInt("123")); *//number*

console.log(typeof parseInt(456)); *//number*

console.log(typeof parseInt(3.14)); *//number. Virgülden sonrası atılır.*

var empty = parseInt(null);

var undefinable = parseInt(undefined);

var str = parseInt("jQuery");

console.log(empty +"=>"+ typeof empty); *//NaN=>number.*

console.log(undefinable +"=>"+ typeof undefinable); *//NaN=>number*

console.log(str +"=>"+ typeof str); *//NaN=>number*

*//(NaN, Not a Number anlamına gelir. Başarılı bir dönüşüm olmadı)*

**‘**parseFloat() ve Number() fonksiyonları : Bu fonksiyonlar parseInt’den farklı olarak hem virgüllü hem de tamsayıları dönüştürebilmektedirler.

**Otomatik Tip Dönüşümü(Auto Convert)**

console.log("55" + 22); *//5522*

*//İşlem önceliği soldan sağa doğru olduğu için önce iki sayı toplanır sonra string*

*//ifadeyle birleştirilir.*

console.log(25 + 20 + "50"); *//4550*

*//Javascript matematiksel işlemlerde string'leri sayılara dönüştürür. Birleştirme*

*//operatörü(+) hariç.*

var x = "100";

var y = "10";

console.log(x - y); *//90*

console.log(x \* y); *//1000*

console.log(x / y); *//10*

**8.Kaçış Karakterleri(Escape Characters)**

\n : Yeni satır.

\t : Tab.

\\ : Backslash.

\’ : Tek tırnak içinde tek tırnak kullanımı.

\” : Çift tırnak içinde çift tırnak kullanımı.

**‘**\n ve \t konsol komutlarıyla beraber kullanılır.

**‘**\\ , \’ ve \” ifadeleri string değerler içerisinde kullanılır.

**9.Yorum Satırları(Comment Lines)**

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

// : Tek satırlık yorumlar için

/\* \*/ : Birden fazla satır için

**10.Operatörler(Operators)**

Javascript’te operatörler diğer programlama dillerinde olduğu gibi kullanılmaktadır. Konuyu uzatmamak adına gereksiz detay vermekten kaçınıyorum.

|  |  |
| --- | --- |
| **Aritmetik Operatörler** | |
| + | String ifadeleri birleştirir, sayısal ifadeleri toplar. |
| - | Çıkarma yapmak için kullanılır. |
| \* | Çarpma yapmak için kullanılır. |
| / | Bölme yapmak için kullanılır. |
| % | Bölümden kalanı bulmak için kullanılır. |

console.log(10 % 4); *//2*

|  |  |
| --- | --- |
| **Atama Operatörleri** | |
| = | Soldaki değere sağdakini atar. |
| += | Topla ve ata. |
| -= | Çıkar ve ata. |
| \*= | Çarp ve ata. |
| /= | Böl ve ata. |
| %= | Mod al ve ata. |

var number = 20;

number += 5;

console.log(number); *//25*

*//(number = number+5) ile (number += 5) aynı işlemi yapmaktadır.*

|  |  |
| --- | --- |
| **Arttırma ve Azaltma Operatörleri** | |
| x++ | x'i bir arttır. |
| ++x | x'i bir arttır. |
| x-- | x'i bir azalt. |
| --x | x'i bir azalt. |

Aralarındaki farkı bir örnek örnek üzerinden açıklayabilirim.

var x = 10;

console.log(x++); *//10*

console.log(x); *//11*

var y = 10;

console.log(++y); *//11*

console.log(y); *//11*

|  |  |
| --- | --- |
| **Karşılaştırma Operatörleri** | |
| == | Eşitse. |
| === | Eşitse.(Veri tipleri de karşılaştırmaya dahil edilir) |
| != | Eşit değilse. |
| !== | Eşit değilse.(Veri tipleri de karşılaştırmaya dahil edilir) |
| > | Büyükse. |
| < | Küçükse. |
| >= | Büyük ya da eşitse. |
| <= | Küçük ya da eşitse. |

console.log(5 != 5); *//false*

console.log(5 != "5"); *//false*

console.log(5 !== "5"); *//true*

|  |  |
| --- | --- |
| **Mantıksal Operatörler** | |
| && | Mantıksal **VE** operatörü. Tüm koşulların sağlanması durumunda **true** aksi halde **false** döner. |
| || | Mantıksal **VEYA** operatörü. Koşullardan en az biri sağlanıyorsa **true**, hiçbir koşulun sağlanmaması durumunda ise **false** döner. |
| ! | Mantıksal **DEĞİL** operatörü. Kendisinden sonra gelen değerin değilini yani tersini(zıttını) döndürür. |

console.log(1 == 1 && 4 < 6); *//true*

console.log(1 == 1 && 6 < 4); *//false*

console.log(1 == 1 && 6 > 4 && 10 < 11); *//true*

console.log(1 == 1 || 2 < 1); *//true*

console.log(1 == 1 || 1 == 10 || 10 < 9 || 3 > 100); *//true*

console.log(0 > 3 || 5 > 10); *//false*

console.log(!false); *//true*

*//0'dan büyük sayılar true olarak kabul edildiği için false değeri döndü.*

console.log(!5); *//false.*

|  |  |
| --- | --- |
| **Bit Düzeyinde İşlem Yapan Operatörler** | |
| & | **AND** operatörü. |
| | | **OR** operatörü. |
| ~ | **NOT** operatörü. |
| ^ | **XOR** operatörü. |

console.log(25 & 14); *//8*

console.log(25 | 14); *//31*

console.log(255 | 0); *//255*

console.log(25 ^ 14); *//23*

console.log(255 ^ 0); *//255*

console.log(~26); *//-27*

*//25 : 00011001 ->25 sayısının iki tabanındaki karşılığı*

*//14 : 00001110 ->14 sayısının iki tabanındaki karşılığı*

*//&*

*//8  : 00001000 ->25 & 14 işleminin sonucu*

*//|*

*//31 : 00011111 ->25 | 14 işleminin sonucu*

*//^*

*//23 : 00010111 ->25 ^ 14 işleminin sonucu*

|  |  |
| --- | --- |
| **Kaydırma Operatörleri** | |
| << | Sola kaydırma. |
| >> | Sağa kaydırma. |

console.log(16 << 1); *//32 (Sayı 2 ile çarpılır)*

console.log(16 << 2); *//64 (Sayı 2 defa 2 ile çarpılır)*

console.log(16 >> 1); *//8 (Sayı 2'ye bölünür)*

console.log(16 >> 2); *//4 (Sayı 2 defa 2'ye bölünür.)*

console.log(17 << 1); *//34*

console.log(17 << 2); *//68*

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

console.log(17 >> 2); *//4*

|  |  |
| --- | --- |
| **Özel Operatörler** | |
| typeof | Bu operatör kendisinden sonra yazılan verinin tipini döndürür. |
| new | Programcı tarafından oluşturulan ya da hazır gelen class’lardan obje üretmek için kullanılır. |
| this | İçinde bulunduğumuz objeyi döndürür. |

Global scope’da this window objesini verir.

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

Obje içerisinde this objenin kendisini verir.

var person = {

    firstName: "Eren",

    age: 27,

    showInfos: function(){

        console.log(this); *//person objesi döner*

        console.log(this.firstName, this.age); *//Eren 27*

    }

}

person.showInfos();

Fonksiyon içerisinde this window objesini verir.

function Student(first\_name, age){

    this.first\_name = first\_name;

    this.age = age;

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

}

Student("Eren", 27);

console.log(first\_name, age); *//Eren 27*

*//Fonksiyon içerisinde this operatörü ile window objesi içinde first\_name ve age adında*

*//iki değişken tanımladık*

Class içerisinde this ilgili class’dan oluşturulan objeyi verir.

function Student(first\_name, age){

    this.first\_name = first\_name;

    this.age = age;

    console.log(this); *//this Student objesini verir.*

    console.log(this.first\_name, this.age); *//Eren 27*

}

var student1 = new Student("Eren", 27);

console.log(first\_name, age); *//Hata meydana gelir.*

*//Class içerisinde this operatörü ile Student objesi içinde first\_name ve age adında iki*

*//özellik tanımladık*

Bir fonksiyon hem class hem de fonksiyon olarak kullanılırsa bu durumda this aşağıdaki örnekte gösterildiği gibi çalışmaktadır.

function Student(first\_name, age){

    this.first\_name = first\_name;

    this.age = age;

    console.log(this);

    console.log(this.first\_name, this.age);

}

*//Fonksiyon çalıştırılır, window objesi içinde ilgili değişkenler tanımlanır ve değerler*

*//konsola yazılır.*

Student("Eren", 27);

*//first\_name ve age obje içinde tanımlanır ve constructor ile aktarılan değerler konsola*

*//yazılır.*

var student1 = new Student("Enis", 30);

*//Sonuç olarak bu değerler birbirleriyle karışmazlar.*

|  |  |
| --- | --- |
| delete | Bu operatör ile objelerin özellik ve metotları silinebilir. |
| instanceof | Bir objenin constructor’larını öğrenmek için kullanılır. |

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*

console.log(Class1 instanceof Object); *//true*

console.log(Class1 instanceof Function); *//true*

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

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

|  |  |
| --- | --- |
| in | Özellik ya da metot belirtilen obje içindeyse ya da prototip zincirinde tanımlıysa true değerini döndürür. |

*//NOT : Özellik, metot isimleri tırnak içinde yazılmalıdır ve küçük-büyük harf ayrımına*

*//dikkat edilmelidir.*

console.log("PI" in Math); *//true*

console.log("pow" in Math); *//true*

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

console.log("Pow" in Math); *//false*

var product = {product\_name:"Apple", stock:10}

console.log("product\_name" in product); *//true*

console.log("stock" in product); *//true*

console.log("toString" in product); *//true*

*//Dizi indeksleri tırnak içine alınmayabilir.*

var days = ["Sunday", "Monday", "Tuesday"];

console.log("Sunday" in days); *//false*

console.log("0" in days); *//true*

console.log(1 in days); *//true*

console.log(4 in days); *//false*

**Ternary Operatörü**

*//(koşul) ? işlem1 : işlem2*

console.log((true) ? "Elma" : 1); *//Elma*

console.log((false) ? "Elma" : 1); *//1*

(2 > 1) ? console.log("Elma") : console.log(1); *//Elma*

**11.Karar Yapıları(Conditional Statements)**

Javascript’te karar yapıları diğer programlama dillerinde olduğu gibi kullanılır. if-else ve switch-case yapılarını kullanabilirsiniz.

**12.Döngüler(Loops)**

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

**‘**for in döngüsü :

var letters = ["a", "b", "c", "d"];

for(var index in letters){

console.log(index); *//0, 1, 2, 3 -> Index numaraları konsola yazıldı.*

}

*//O halde dizi içeriğine şu şekilde erişebiliriz*

for(var index in letters){

console.log(letters[index]); *//a, b, c, d*

}

*//for in ile objeler üzerinde gezinebilirsiniz*

var city = {name:"Kocaeli", code:41, population:1997258}

for(var key in city){

console.log(key); *//name, code, population*

}

for(var key in city){

console.log(city[key]); *//Kocaeli, 41, 1997258*

}

*//for in ile String ifadenin karakterlerini okuyabilirsiniz*

var data = "eren\_sati@hotmail.com"

for(var index in data){

if(index == 4){

break;

}

console.log(data[index]); *//e,r,e,n*

}

**‘**for of döngüsü : **ECMAScript 6** ile birlikte gelen yeni bir döngüdür. [Edge+]

*//for of ile diziler üzerinde gezinmek*

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

for(var language of languages){

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

}

*//for of ile string'ler üzerinde gezinmek*

var fruit = "Melon";

for(var char of fruit){

    console.log(char); *//M, e, l, o, n*

}

for of ile objeler üzerinde **gezinemiyoruz**.

**13.Deyimler**

**‘**break : Döngüyü sonlandırmak için kullanılır.

**‘**continue : Döngü içinde belirli bir koşulun kontrolüyle bir sonraki döngü adımına gidilmesini gerçekleştirir.

**‘**return : Fonksiyonu sonlandırmak için kullanılır. return [data] şeklinde kullanılırsa fonksiyonun çağrıldığı yere ilgili değeri döndürür. **return** ifadesinden sonra yazılan hiçbir komut çalıştırılmaz.

**14.İşlem Önceliği**

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

**15.Fonksiyonlar(Functions)**

Javascript’te fonksiyonlar aşağıda gösterildiği gibi farklı yöntemlerle tanımlanabilirler.

*//1*

function function1(){

}

*//2*

var function2 = function(){

}

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

(function(){

})()

**‘**İç içe fonksiyon tanımlanabilir.

**‘**Fonksiyonlar return deyimi ile geriye değer gönderebilir.

function square(x){

    return x \* x;

}

function cube(x){

    return x \* x \* x;

}

console.log(cube(square(2))); *//64*

**‘**Fonksiyonlar tanımlanırken parametre belirtilmese bile çağrılırken parametre gönderilebilir.

function numbers(){

    for(number in arguments){

        console.log(arguments[number]); *//10, 20, 30*

    }

    console.log(arguments.length); *//3*

}

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

*//erişebilirsiniz.*

numbers(10, 20, 30);

**‘**Fonksiyon parametrelerine default değerler verebilirsiniz.

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.

function test\_function(){

    console.log(":)");

}

var function1 = test\_function;

console.log(function1); *//function test\_function(){console.log(":)");}*

function1(); *//:)*

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

var obj1 = {number1: 10, number2: 20}

var obj2 = {number1: 30, number2: 40}

*//this anahtar kelimesi normalde window objesi içindeki number1 ve number2 değişkenlerini*

*//gösterir.* *Ancak apply ve call sayesinde artık obj1 ve obj2 içerisindeki değişkenleri*

*//gösteriyor.*

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 objeyle*

*//kullanılmasını sağlar.*

var copy\_function1 = addNumbers.bind(obj1);

var copy\_function2 = addNumbers.bind(obj2);

copy\_function1(100, 200); *//330*

copy\_function2(100, 200); *//370*

**16.Global Fonksiyonlar ve Global Özellikler**

Global fonksiyon ve özellikler window objesi altında bulunur.

**Global Fonksiyonlar**

Daha önce gördüğümüz String(), Number(), parseInt() ve parseFloat() fonksiyonları birer global fonksiyondur. Bu fonksiyonlar bu başlık altında tekrar incelenmeyecektir.

**‘**isNaN() : Parametre olarak verilen değer sayı ise false, sayı değilse true döner.

console.log(isNaN(100)); *//false*

console.log(isNaN(0)); *//false*

console.log(isNaN("-1")); *//false*

console.log(isNaN("Hello")); *//true*

console.log(isNaN("2005/12/12")); *//true*

console.log(isNaN(NaN)); *//true*

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

console.log(isNaN(-Infinity)); *//false*

**‘**eval() : Parametre olarak verilen string ifadeyi Javascript koduymuş gibi çalıştırır.

console.log(eval("2 + 2")); *//4*

var number = 10;

console.log(eval("number + 10")); *//20*

**‘**isFinite() : Bir sayının sonlu ve yasal bir sayı olup olmadığını kontrol etmek için kullanılır. Parametre olarak verilen sayı NaN, Infinity ya da -Infinity ise **false** aksi halde **true** döner.

console.log(isFinite(10)); *//true*

console.log(isFinite(-10.5)); *//true*

console.log(isFinite(NaN)); *//false*

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

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

console.log(isFinite("Javascript")); *//false*

**Global Özellikler**

**‘**NaN : Bir verinin sayı olmadığını belirten bir değerdir.

*//Özelliğin değeri yine NaN'dır :)*

console.log(NaN); *//NaN*

*//Ancak tipi number'dır.*

console.log(typeof NaN); *//number*

*//Matematiksel işlemlerde kullanıldığında sonuç yine NaN olur.*

var x = 100 / "Apple";

console.log(x); *//NaN*

console.log(x + 10); *//NaN*

*//NaN değer olarak kullanılabilmektedir.*

var y = NaN;

console.log(y); *//NaN*

**‘**Infinity veya -Infinity : Javascript’te pozitif veya negatif sayı sınırının aşıldığını belirten bir değerdir. Sınırı aşan bir sayı Infinity veya -Infinity ile gösterilir.

*//Özelliğin değeri yine Infinity'dir :).*

console.log(Infinity); *//Infinity*

*//Ancak tipi number'dır.*

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

*//Ayrıca 0'a bölme işleminde geriye Infinity değeri dönmektedir.*

console.log(10 / 0); *//Infinity*

console.log(-10 / 0); *//-Infinity*

*//Infinity değer olarak kullanılabilmektedir.*

var x = Infinity;

console.log(x); *//Infinity*

*//Javascript'te pozitif ve negatif sayı sınırını görmek için Number objesinin*

*//MAX\_VALUE ve MIN\_VALUE özelliklerini kullanabilirsiniz.*

console.log(Number.MAX\_VALUE); *//1.79e+308*

console.log(Number.MIN\_VALUE); *//5e-324*

*//Şimdi daha büyük bir sayı kullanıp sınırı aşacağız.*

console.log(1.79e+308); *//1.79e+308*

console.log(1.79e+309); *//Infinity*

**17.Diziler(Arrays)**

Javascript’te diziler aşağıda gösterildiği gibi farklı yöntemlerle tanımlanabilmektedirler.

*//Dizi tanımlama yöntemleri*

*//1.Condensed dizi*

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

*//2.Literal dizi*

let numbers = [1, 2, 3, 4];

*//3.Regular dizi*

let days = new Array();

days[0] = "Sunday";

days[1] = "Monday";

*//Hata meydana gelmez. Ancak 1. index'den 5'e atladığımız için 2, 3 ve 4. index'ler*

*//otomatikmen boş olarak tanımlanır. Bu yüzden dizi 6 elemanlı olur.*

days[5] = "Friday";

console.log(days.length); *//6;*

console.log(days); *//["Sunday", "Monday", empty x 3, "Friday"]*

**‘**Çok boyutlu diziler oluşturabilirsiniz.

let arr1 = Array(new Array(1, 2, 3), new Array(10,20,30,40), new Array(100, 200));

let arr2 = [[1, 2, 3], ["a", "b"], ["+", "-", "\*", "/"]];

let arr3 = new Array();

arr3[0] = new Array(1, 2, 3);

arr3[1] = new Array(10, 20, 30, 40);

**Array Sınıfının Özellik ve Metotları**

*//Özellik ve metotları konsol penceresinde görebiliriz.*

console.log(new Array());

**‘**length : Dizinin eleman sayısını almak ya da değiştirmek için kullanılır.

var numbers = [1, 2, 3, 4];

console.log(numbers.length); *//4*

numbers.length = 2; *//Eleman sayısını 2 olarak değiştirdik. Diğer elemanlar silindi*

console.log(numbers.length); *//2*

console.log(numbers); *//[1, 2]*

**‘**push() : Dizinin sonuna eleman eklemek için kullanılır.

var 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.

var 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 elemanlarını string olarak döndürür.

var numbers = [1, 2, 3];

var str = numbers.toString();

console.log(str); *//1,2,3*

console.log(typeof str); *//string*

**‘**concat() : Dizileri birleştirip geri döndürür.

var arr1 = [1, 2, 3];

var arr2 = [4, 5, 6];

var arr3 = [7, 8, 9];

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

**‘**join() : Parametre olarak verilen değeri dizi elemanları arasına ekler ve string olarak döndürür. Parametresiz kullanımda elemanlar arasına virgül eklenir.

var numbers = [1, 2, 3];

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

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

console.log(numbers.join("asd")); *//1asd2asd3*

**‘**reverse() : Dizi elemanlarını tersten sıralar.

var numbers = [1, 2, 3];

numbers.reverse();

console.log(numbers); *//[3, 2, 1]*

**‘**indexOf() : Parametre olarak verilen değeri dizi içerisinde arar, bulursa ilk geçtiği yerin index numarasını bulamazsa -1 döndürür. İkinci parametrenin kullanımı isteğe bağlıdır, başlangıç index’ini belirtmek için kullanılır.

var cities = ["İstanbul", "İzmir", "Edirne", "Denizli", "İstanbul"];

console.log(cities.indexOf("İstanbul")); *//0*

console.log(cities.indexOf("Eskişehir")); *//-1*

console.log(cities.indexOf("İstanbul", 1)); *//4*

**‘**lastIndexOf() : Parametre olarak verilen değeri dizi içerisinde arar, bulursa son geçtiği yerin indeks numarasını bulamazsa -1 döndürür. İkinci parametrenin kullanımı isteğe bağlıdır, bitiş indeksini belirtmek için kullanılır.

var cities = ["İstanbul", "İstanbul", "Edirne", "Denizli", "İstanbul"];

console.log(cities.lastIndexOf("İstanbul")); *//4*

console.log(cities.lastIndexOf("İstanbul", 3)); *//1*

console.log(cities.lastIndexOf("İstanbul", 4)); *//4*

**‘**slice() : Dizinin bir bölümünü(ya da tamamını) kopyalar ve yeni bir dizi olarak döndürür. İlk parametre başlangıç, ikinci parametre bitiş indeksidir. Bitiş indeksi belirtilmezse dizinin sonuna kadar seçilir. Bitiş indeksi negatif değer aldığında dizi sonundan bitiş indeksi kadar eleman seçilmez.

var 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]*

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

**‘**splice() : Dizi içinden eleman silmek ya da yeni elemanlar eklemek için kullanılır.

*//Elemanları silerken yeni eleman eklemek*

var numbers = [1, 2, 3, 4, 5, 6];

*//splice(startIndex, count, newItems)*

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

var deleted\_items = numbers.splice(2, 2, "A", "B", "C");

console.log(deleted\_items); *//[3, 4]*

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

*//Eleman eklemek*

var numbers = [1, 2, 3, 4];

numbers.splice(2, 0, "A", "B", "C");

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

*//Eleman silmek*

numbers.splice(2, 1);

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

*//Silme işlemi sonrasında ilgili index boş değer içermez. Elemanlar kaydırılır ve*

*//2. index'de "A" yerine "B" değeri bulunur.*

**‘**map() : Dizi elemanlarını belirtilen bir fonksiyondan geçirerek dizi halinde geri döndürür. Orijinal dizi üzerinde değişiklik yapılmaz.

var 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. Parametre*

*//gönderilmezse this, undefined olur.*

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

return item \* item + this;

},"Apple"));

**‘**filter() : Dizi elemanlarını filtreleyip dizi halinde geri döndürür. Orijinal dizi üzerinde değişiklik yapılmaz. map() metodunda olduğu gibi iki ya da üç parametreli olarak kullanabilirsiniz. Fonksiyon bloğundan sonra kendi parametrenizi gönderebilirsiniz.

var 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 değerini döndürür. map() metodunda olduğu gibi iki ya da üç parametreli olarak kullanabilirsiniz. Fonksiyon bloğundan sonra kendi parametrenizi gönderebilirsiniz.

var numbers = [2, 4, 6, 8];

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

return item <= 8;

}));

console.log(numbers.every(function(item){ *//false -> Çünkü dizide 6’dan büyük sayı var.*

return item <= 6;

}));

**‘**some() : every() gibi çalışır fakat elemanlardan en az biri koşulu sağlıyorsa true aksi halde false döner. İki ya da üç parametreli olarak kullanılabilir. Fonksiyon bloğundan sonra özel parametre gönderilebilir.

var numbers = [2, 4, 6, 8];

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

return item <= 2;

}));

console.log(numbers.some(function(item){ *//false -> Çünkü dizide 8’den büyük sayı yok.*

return item > 8;

}));

**‘**forEach() : Dizideki her eleman için bir fonksiyon çalıştırır. İki ya da üç parametreli olarak kullanılabilir. Fonksiyon bloğundan sonra özel parametre gönderilebilir.

var numbers = [2, 4, 6, 8];

numbers.forEach(function(item){

console.log(item \* item); *//4, 16, 36, 64 değerleri sırasıyla konsola yazılır.*

});

function x10(item, index, arr){

arr[index] = item \* 10;

}

numbers.forEach(x10);

console.log(numbers); *//[20, 40, 60, 80]*

**18.Hata Yönetimi**

Javascript’te hata yönetimi try – catch – finally blokları ile yapılır.

try{

*//Hata verebilecek kodlar buraya yazılır.*

}

catch(err){

*//Hata meydana gelirse catch blogu çalışır.*

*//Browser tarafından bir Error objesi fırlatılır. Bu objeyi catch bloguna verdiğimiz*

*//parametre(err) ile yakalıyoruz.*

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

*//hata hakkında bilgi verir.*

}

finally{

*//Hata olsun olmasın çalışması istenen kodlar varsa buraya yazılacak.*

}

Örnek :

try{

*//y adında bir değişken yok. Bu satır try bloğu içinde olmasaydı Error objesi browser*

*//tarafından yakalanıp konsola yazılırdı.* *Ama artık bizim kontrolümüzde. Browser bu*

*//hatadan etkilenmeyecek.*

var x = y + 10;

}

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 5 farklı değerden birini alır : RangeError, ReferenceError, SyntaxError, TypeError, URIError

**‘**Error objesinin message özelliği gerçekleşen hatanın açıklamasını gösteren bir string’dir. Açıklama metni browser’a bağlı olarak değişir.

**‘**Kendi Error objelerinizi oluşturabilir ve bunları throw anahtar kelimesiyle fırlatabilirsiniz.

var my\_error = new Error();

my\_error.name = "Test ErrorName";

my\_error.message = "Test ErrorMessage";

try{

throw my\_error;

}

catch(err){

console.log(err); *//Test ErrorName: Test ErrorMessage*

console.log(err.name); *//Test ErrorName*

console.log(err.message); *//Test ErrorMessage*

}

throw anahtar kelimesiyle fırlattığınız hata string, number, boolean veya object tipinde olabilir.

**19.ECMAScript 6+**

**Arrow Functions**

Artık fonksiyonları daha kısa bir sözdizimle tanımlayabiliyoruz. Aşağıdaki örnekler herşeyi açıklıyor. [Edge+]

*//Standart sözdizim*

var writeMessage = function(){

     console.log(":)");

}

*//Arrow function*

var writeMessage = () => {

    console.log(":)");

}

var square = (number) => {

    return number \* number;

}

*//Fonksiyonun tek parametresi varsa parantez kullanılmayabilir*

var square = number => {

    return number \* number;

}

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

var square = number => return number \* number;

*//Fonksiyon tek bir işlem yapıyorsa ve return ile geriye değer gönderiyorsa return'a da*

*//gerek yoktur.*

var square = number => number \* number;

**Template Literal**

Kullanımını bir örnek üzerinden açıklayacağım. [Edge+]

var product = {

    product\_name: "Samsung A70",

    price: 4000,

    stock: 10

}

var product\_infos =

"Product Name : " + product.product\_name +

" Price : " + product.price +

" Stock : " + product.stock;

console.log(product\_infos);

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

var product\_infos =

`Product Name : ${product.product\_name} Price : ${product.price} Stock : ${product.stock}`;

**‘**Template literal içerisinde Javascript kodları kullanabilirsiniz.

var firstName = "Eren";

var age = 27;

var city = "Kocaeli";

var message = () => "Merhaba";

var html = `

  <ul>

    <li>${firstName}</li>

    <li>${age}</li>

    <li>${city}</li>

    <li>${10 / 4}</li>

    <li>${message()} Eren</li>

  </ul>

`;

document.body.innerHTML = html;

**Destructing**

[Edge 14+]

*//Dizilerden değişkenlere değer aktarmak*

var number1, number2;

var numbers = [1, 2, 3, 4];

[number1, number2] = numbers;

console.log(number1); *//1*

console.log(number2); *//2*

*//Aynı işlem değişkenin tanımlandığı satırda da yapılabilir.*

var letters = ["a", "b", "c", "d"];

var [letter1, letter2] = letters;

console.log(letter1); *//a*

console.log(letter2); *//b*

*//Objelerden değişkenlere değer aktarmak*

var numbers = {

    a: 1,

    b: 2,

    c: 3,

    d: 4

};

var {a, c} = numbers; *//Değişken adlarının aynı olması gerekiyor.*

console.log(a); *//1*

console.log(c); *//3*

*//Değişken adlarının farklı olmasını istiyorsanız tanımlamayı aşağıdaki gibi*

*//yapabilirsiniz.*

var {b: number1, c: number2, d: number3} = numbers;

console.log(number1); *//2*

console.log(number2); *//3*

console.log(number3); *//4*

**Spread Operator**

[Edge+]

*//Yaptığı işlem : "langs[0], langs[1], langs[2], langs[3]" şeklinde dizinin her*

*//elemanını virgül ile ayırıp almak.*

var langs = ["C#", "Java", "Python", "C++"];

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

*//Örnek kullanım*

var langs2 = ["PHP", "ASP", ...langs]

console.log(...langs2); *//PHP ASP C# Java Python C++*

*//Örnek kullanım*

var addNumbers = (x, y, z) => x + y + z;

var numbers = [1, 2, 3];

console.log(addNumbers(...numbers)); *//6*

**ES6+ OOP Yapısı**

*//Bir ES6 class'ının yapısı*

class Person{

*//Özelliklere varsayılan değerler verebilirsiniz*

    firstName;

    age;

*//constructor içinde tanımlanan özellik ve metotlar obje içinde oluşturulur.*

    constructor(firstName, age){

        var \_id = "1"; *//private özellik. sadece constructor içinde kullanılabilir.*

        this.firstName = firstName;

        this.age = age;

        this.showInfos = function(){

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

        }

        Person.Count++;

    }

*//Bu bölgede tanımlanan metotlar objenin prototipi içinde oluşturulur.*

    testFunction(){

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

    }

*//static bir özelliktir. obje üzerinden kullanılamaz. "Person.Count" şeklinde*

*//kullanabilirsiniz.*

    static Count = 0;

*//static bir metottur. obje üzerinden kullanılamaz. "Person.UpdateCount()" şeklinde*

*//kullanabilirsiniz.*

    static UpdateCount(){

        this.Count--;

    }

}

var person1 = new Person("Eren", 27);

var person2 = new Person("Enes", 30);

console.log(person1);

console.log(person2);

console.log(Person.Count); *//2*

person1.showInfos(); *//Eren 27 1*

person1.testFunction(); *//Eren 27*

Kalıtım :

class Person{ *//base class*

    constructor(firstName, age){

        this.firstName = firstName;

        this.age = age;

    }

    showInfos(){

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

    }

}

class Employee extends Person{

    constructor(firstName, age, salary){

        super(firstName, age); *//base class'ın constructor'ını çağırıyoruz.*

        this.salary = salary;

    }

*//showInfos metodunu override ediyoruz.*

    showInfos(){

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

*//Bu class içinden base class'daki bir metodu kullanmak istersek super anahtar*

*//kelimesinden yararlanıyoruz.*

        super.showInfos();

    }

}

var employee = new Employee("Eren", 27, 4000);

console.log(employee);

employee.showInfos();

**20.Callback Fonksiyonlar**

Öncelikle javascript kodlarının asenkron olarak çalıştığını bir örnek üzerinden görelim.

function Process1(){

    setTimeout(function(){

        console.log("Process 1");

    },2000)

}

function Process2(){

    console.log("Process 2");

}

*//Önce Process2 daha sonra Process1 çalışır.*

*//Yani Process2 Process1'i beklemez.*

Process1();

Process2();

Bu noktada amacımız Process1 fonksiyonu işini bitirdikten sonra Process2’nin çalışmasını sağlamak. Yani senkron bir yapı kurmamız gerekiyor. Bu durumda callback fonksiyonlardan yararlanıyoruz.

function Process1(callback){

    setTimeout(function(){

        console.log("Process 1");

        callback();

    },2000)

}

function Process2(){

    console.log("Process 2");

}

Process1(Process2);

**21.OOP(Object Oriented Programming)**

Javascript’te OOP yapısı oluşturulurken fonksiyon veya class’lardan yararlanılır.

Obje oluşturmanın üç farklı yolu vardır.

**‘Object Literal** sözdizimi ile obje oluşturabilirsiniz.

var product = {

    product\_name: "Samsung A70",

    stock: 10,

    show\_infos:function(){

        console.log(this.product\_name, this.stock);

    }

};

*//Özellik ve metotlar üzerinde değişiklik yapabilirsiniz.*

product.product\_name = "Samsung A50";

product.price = 4000; *//Yeni özellik ve metotlar ekleyebilirsiniz.*

*//Özellik ve metotları delete operatörü ile silebilirsiniz.*

delete product.show\_infos;

delete product.stock;

console.log(product); *//{product\_name: "Samsung A50", price: 4000}*

**‘**Object sınıfı üzerinden obje oluşturabilirsiniz.

*//Object Literal sözdiziminden hiçbir farkı yoktur.*

var person = new Object();

person.first\_name = "Eren",

person.age = 27;

**‘**Bir class veya fonksiyon üzerinden obje oluşturabilirsiniz.

function Student(first\_name, age){

    this.first\_name = first\_name;

    this.age = age;

}

var student = new Student("Eren", 27);

**‘**Javascript'te tüm class’lar Object sınıfının mirasçısıdır ve tüm objelerin prototype ve constructor özellikleri bulunmaktadır.

constructor özelliği objelerin constructor’ını geriye döndürür.

\_\_proto\_\_ özelliği objelerin prototipini geriye döndürür.

var obj1 = {};

var obj2 = new Object();

var Class1 = function(){}

var obj3 = new Class1();

*//obj1 ve obj2 Object sınfının constructor'ını kullanır.*

*//obj3 için kendimiz bir constructor tanımladık.Yani Class1 fonksiyonu constructor’ımız.*

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

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

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

var obj1 = {};

var obj2 = new Object();

var Class1 = function(){}

var obj3 = new Class1();

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

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

console.log(obj3.\_\_proto\_\_); *//Class'ımızın kendi prototipi döner.*

**‘**Class’larımızın prototipleri içerisinde başlangıçta herhangi bir metot ve özellik yoktur. Ancak Object sınıfının prototipi geliştiriciler tarafından tanımlanan bazı özellik ve metotlarla birlikte gelmektedir. Class’larımızın prototipleri, Object sınıfının prototipini de içerisinde barındırmaktadır çünkü Javascript'te tüm class’lar Object sınıfının mirasçısıdır. Dolayısıyla Object sınıfı kaynaklı prototipin içindeki özellik ve metotları kendi objelerimiz üzerinde kullanabiliriz. Hatta bu metot ve özellikleri sınıflarımız içerisinde override edebiliriz.

*//Object sınıfı kaynaklı toString() metodunun override edilmesi*

function Student(first\_name, age){

    this.first\_name = first\_name;

    this.age = age;

    this.toString = function(){

      console.log("deneme yazısı...")

    }

}

var student = new Student("Eren", 27);

student.toString(); *//deneme yazısı...*

*//obje içerisi > objenin prototipi > Object sınıfının prototipi : Bu sıralamaya göre*

*//toString() metodu ilk olarak nerede tanımlanmışsa orada çalıştırılır.*

**‘**Objelerinizin prototipleri içerisinde özellik ve metotlar tanımlayabilirsiniz.

function Employee(first\_name, age){

    this.first\_name  = first\_name;

    this.age = age;

}

Employee.prototype.show\_infos = function(){

    console.log("Name : " + this.first\_name + " - Age : " + this.age);

}

var employee1 = new Employee("Eren", 27);

employee1.show\_infos(); *//Name : Eren - Age : 27*

console.log(employee1);

*//Bu metot class içinde tanımlansaydı her obje için bellekte 1 kopyası oluşturulacaktı.*

*//Şuan bellekte sadece 1 örneği bulunuyor ve tüm objeler ortak olarak* *onu kullanabiliyor.*

**‘**Bir objenin prototipini başka bir objenin içeriğinden oluşturabilirsiniz.

var obj1 = {

    show\_infos: function(){}

}

*//Boş bir obje oluşturur ve bu objenin prototipi içerisinde show\_infos metodu olur.*

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

var obj1 = {

    show\_infos: function(){}

}

var obj2 = Object.create(obj1);

console.log(obj2);

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

function Person(first\_name, age){

    this.first\_name = first\_name;

    this.age = age;

}

Person.prototype.show\_infos = function(){

  console.log(this.first\_name + "-" + this.age);

}

function Employee(first\_name, age){

  this.first\_name = first\_name;

  this.age = age;

}

Employee.prototype.Count = 0; *//Bu özellik birazdan silinecek :)*

Employee.prototype = Person.prototype;

var employee1 = new Employee("Eren", 27);

console.log(employee1);

Ancak bu örnekte Employee objesinin kendi prototipi olmaz. Çünkü kendi prototipi Person sınıfının prototipini göstermektedir. Eğer Employee objesinin kendi prototipi olmasını ve içerisinde de Person objesinin prototipini barındırmasını istiyorsanız

Employee.prototype = Person.prototype;

yerine

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

kullanmanız gerekir. Bu sayede Employee objesine özel metot ve özellikleri kendi prototipinde tanımlayabiliriz.

Konuyu fonksiyon tabanlı basit bir kalıtım örneği yaparak bitiriyorum :)

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("Eren", 27, 4000);

console.log(employee);

employee.showInfos(); *//Eren 27 4000*

**22.Javascript Objeleri**

String **Objesi**

**Özellikler**

prototype

constructor

length : String ifadenin uzunluğunu verir.

console.log("Eren".length); *//4*

**Metotlar**

charAt(index) : String ifade içinde belirtilen konumdaki karakteri verir. İlgili index yoksa boş string döner.

var str = "Javascript";

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

charCodeAt(index) : String ifade içinde belirtilen konumdaki karakterin Unicode değerini verir. Büyük küçük harf duyarlılığı vardır.

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

[string].concat(string2, string3, ..., stringX) : String ifadeleri birleştirip geri döndürür.

console.log("TP-".concat("LINK")); *//TP-LINK*

String.fromCharCode() : Girilen Unicode değerinin karakterini döndürür.

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

indexOf() : Parametre olarak verilen değeri string ifadenin içinde arar, bulursa ilk geçtiği yerin index numarasını bulamazsa -1 döndürür. İkinci parametrenin kullanımı isteğe bağlıdır, başlangıç index’ini belirtmek için kullanılır.

var first\_name = "Zeynep";

console.log(first\_name.indexOf("e")); *//1*

console.log(first\_name.indexOf("e", 1)); *//1*

console.log(first\_name.indexOf("e", 2)); *//4*

console.log(first\_name.indexOf("nep")); *//3*

console.log(first\_name.indexOf("a")) *//-1*

lastIndexOf() : Parametre olarak verilen değeri string ifadenin içinde arar, bulursa son geçtiği yerin index numarasını bulamazsa -1 döndürür. İkinci parametrenin kullanımı isteğe bağlıdır, bitiş index’ini belirtmek için kullanılır.

var first\_name = "Zeynep";

console.log(first\_name.lastIndexOf("e")); *//4*

console.log(first\_name.lastIndexOf("e",1)); *//1*

console.log(first\_name.lastIndexOf("e",2)); *//1*

console.log(first\_name.lastIndexOf("e",3)); *//1*

console.log(first\_name.lastIndexOf("e",4)); *//4*

replace() : String ifade içinde değişiklik yapıp geri döndürür.

var question = "C# mı Java mı?";

*//Değişiklik sadece ilk "mı" için yapılır.*

console.log(question.replace("mı", "mi")); *//C# mi Java mı?*

console.log(question); *//C# mı Java mı? -> Orjinal veri değişiklikten etkilenmez.*

toLowerCase() , toUpperCase() : Harfleri küçültüp(ya da büyütüp) geri döndürür.

console.log("istanbul".toUpperCase()); *//ISTANBUL -> i harfi I'ya dönüştü.*

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

toLocaleLowerCase() , toLocaleUpperCase() : Tarayıcının diline göre küçük-büyük harf dönüşümü yapar.

console.log("istanbul".toLocaleUpperCase()); *//İSTANBUL*

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

substr() : String ifade içinde belirtilen konumdan belli sayıda karakteri getirir.

var msg = "Dokuz at bir kazığa bağlanmaz.";

console.log(msg.substr(6, 2)); *//at*

console.log(msg.substr(1)); *//okuz at bir kazığa bağlanmaz.*

console.log(msg.substr(-3)); *//az.*

console.log(msg.substr(-4, 2)); *//ma*

substring() : String ifade içinde belirtilen iki konum arasındaki karakterleri getirir.

var 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 ayarlanabilir.

var telephone\_number = "0553 029 77 85";

console.log(telephone\_number.split(" ")); *//[0553, 029, 77, 85]*

console.log(telephone\_number.split(" ", 3)); *//[0553, 029, 77]*

console.log(telephone\_number.split("", 4)); *//[0, 5, 5, 3]*

var website = "www.erensati.com";

console.log(website.split(".")); *//[www, erensati, com]*

toString() , valueOf() : String ifadeyi döndürür.

var lang = "C#";

console.log(lang.toString()); *//C#*

console.log(lang.valueOf()); *//C#*

Number **Objesi**

**Özellikler**

prototype

constructor

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

console.log(Number.MAX\_VALUE); *//1.79e+308*

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

console.log(Number.MIN\_VALUE); *//5e-324*

NaN

console.log(Number.NaN); *//NaN*

**Metotlar**

toFixed() : Virgülden sonraki basamak sayısını ayarlayıp string olarak geri döndürür.

var number = 367.32342;

console.log(number.toFixed(1)); *//367.3*

console.log(number.toFixed(2)); *//367.32*

console.log(number.toFixed(3)); *//367.323*

toString() : Sayıyı string olarak döndürür. Parametre kullanımı isteğe bağlıdır ve sayı tabanını ayarlamak için kullanılır. Parametre 2 ile 36 aralığında bir tamsayı olmalıdır.

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*

valueOf() : Sayıyı döndürür.

Error **Objesi**

**Özellikler**

name

message

Yukarıdaki özellikleri **Hata Yönetimi** başlığı altında zaten görmüştük.

Boolean **Objesi**

**Özellikler**

prototype

constructor

**Metotlar**

toString()

valueOf()

Math **Objesi**

Math objesinin constructor’ı olmadığı için new anahtar kelimesiyle obje oluşturulamaz.

**Özellikler**

E : Eular sayısını döndürür.

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

PI : PI sayısını döndürür.

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

LN2 : 2’nin logaritmasını döndürür.

console.log(Math.LN2); *//0.6931471805599453*

LN10 : 10’un logaritmasını döndürür.

console.log(Math.LN10); *//2.302585092994046*

SQRT1\_2 : ½’nin karekökünü döndürür.

console.log(Math.SQRT1\_2); *//0.7071067811865476*

SQRT2 : 2’nin karekökünü döndürür.

console.log(Math.SQRT2); *//1.4142135623730951*

**Metotlar**

abs() : Parametre olarak verilen sayının mutlak değerini döndürür.

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.

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.

console.log(Math.pow(3, 2)); *//9*

console.log(Math.pow(3, 3)); *//27*

console.log(Math.pow(3, 4)); *//81*

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

console.log(Math.pow(3.6, 2)); *//12.96*

ceil() : Verilen virgüllü sayıyı yukarı yuvarlar.

floor() : Verilen virgüllü sayıyı aşağı yuvarlar.

console.log(Math.ceil(6.3)); *//7*

console.log(Math.ceil(-2.1)); *//-2*

console.log(Math.floor(6.9)); *//6*

console.log(Math.floor(-6.1)); *//-7*

console.log(Math.ceil(8.669)); *//9*

console.log(Math.floor(4.999)); *//4*

console.log(Math.ceil(6)); *//6*

console.log(Math.floor(-9)); *//-9*

round() : Verilen virgüllü sayıyı yakın olana yuvarlar. 0.5, -2.5, 4.5 gibi bir sonraki ve bir önceki tamsayıya eşit uzaklıktaki değerler yukarı yuvarlanır.

console.log(Math.round(0.6)); *//1*

console.log(Math.round(0.4)); *//0*

console.log(Math.round(3.5)); *//4*

console.log(Math.round(-2.1)); *//-2*

console.log(Math.round(-2.7)); *//-3*

console.log(Math.round(-2.5)); *//-2*

max() : Parametre olarak verilen sayıların en büyüğünü döndürür.

min() : Parametre olarak verilen sayıların en küçüğünü döndürür.

console.log(Math.max(2, 3, 4, 5, 1, 10, 2000, 30)); *//2000*

console.log(Math.min(2, 3, 4, 5, 1, 10, 2000, 30)); *//1*

console.log(Math.max(-5, -10, -50)); *//-5*

console.log(Math.min(-2.2, -2.6)); *//-2.6*

random() : 0 ile 1 arasında rastgele sayı üretir.

console.log(Math.random()); *//0.07137295453298264*

console.log(Math.random()); *//0.5380105121592655*

console.log(Math.random()); *//0.7274145914663346*

Görüldüğü gibi her seferinde farklı bir değer elde edilmekte.

Random fonksiyonunu farklı amaçlarla kullanabilirsiniz. Örneğin 0 ile 100 arasında(0 ve 100 dahil) bir tamsayı üretmek için random fonksiyonu 100 ile çarpılıp daha sonra yuvarlama fonksiyonlarından biri ile sayının ondalık kısmı atılabilir.

console.log(Math.random() \* 100); *//79.1022032978974*

console.log(Math.round(Math.random() \* 100)); *//92*

Benzer şekilde 0 ile 10(0 ve 10 dahil) arasında bir tamsayı üretmek için random fonksiyonu 10’la, 0 ile 1000 arasında bir tamsayı üretmek için random fonksiyonu 1000 ile çarpılabilir.

Date **Objesi**

**Özellikler**

prototype

constructor

**Metotlar**

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). Pazar : 0, Pazartesi : 1, …

Aşağıdaki örnek 12/12/2015 tarihinde yapılmıştır. Konsola yazılan değerler sistem tarihine bağlıdır. Bilgisayarınızın tarihini değiştirip kodu yeniden çalıştırırsanız konsola yazılan değerler değişir.

var date = new Date();

console.log(date.getDate()); *//12*

console.log(date.getMonth()); *//11*

console.log(date.getFullYear()); *//2015*

console.log(date.getDay()); *//6 -> Cumartesi*

Aşağıdaki örnekte ise gün, ay, yıl bilgilerini Date objesine parametre olarak verdiğimiz tarihten alıyoruz.

var date = new Date("May 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 ise opsiyoneldir.

var date = new Date();

console.log(date.getDate(), date.getMonth(), date.getFullYear()); *//12, 11, 2015*

date.setDate(1);

date.setMonth(0);

date.setFullYear(1999);

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

Aşağıdaki örnek yapılırken saat 12:03:56’yı göstermekteydi. Konsola yazılan değerler sistem saatine bağlıdır.

var date = new Date();

console.log(date.getHours()); *//12*

console.log(date.getMinutes()); *//3*

console.log(date.getSeconds()); *//56*

console.log(date.getMilliseconds()); *//462*

Aşağıdaki örnekte ise saat, dakika, saniye, milisaniye bilgilerini Date objesine parametre olarak verdiğimiz zamandan alıyoruz.

var date = new Date("16 May, 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.

var date = new Date("16 May, 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()

valueOf()

**23.Promise Objesi – async, await**

Bir promise objesi tanımlamak için Promise construtor’ı kullanılır.

var promise = new Promise(function(resolve, reject){});

console.log(promise);

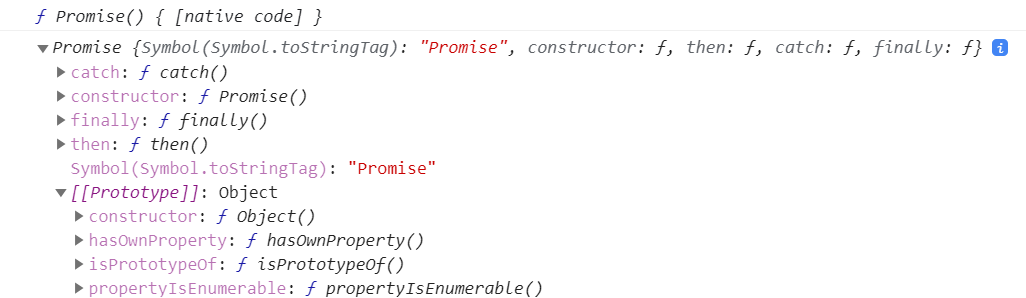


Promise objesi hakkında daha fazla bilgi edinmek için constructor ve \_\_proto\_\_ özelliklerine bakabiliriz.

var promise = new Promise(function(resolve, reject){});

console.log(promise.constructor);

console.log(promise.\_\_proto\_\_);



Promise objesi, içerisinde birtakım işlemler yapıp geriye olumlu ya da olumsuz iki farklı sonuç dönebileceğimiz bir objedir. Olumlu sonuç dönmek için resolve, olumsuz sonuç dönmek için reject fonksiyonları kullanılır.

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 :

var promise = new Promise(function(resolve, reject){

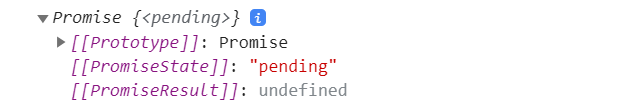
    setTimeout(function(){

        resolve("Merhaba"); *//Parametre vererek geriye değer gönderebiliyoruz*

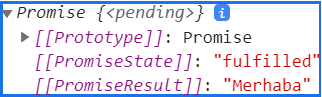
    },2000)

});

console.log(promise);



İki saniye sonra konsol çıktısına tekrar baktığımızda,



State özelliği resolved veya fullfilled, result özelliği ise geriye döndüğümüz değer olacaktır. Bu örnekte resolve fonksiyonunu kullandık ancak aşağıdaki gibi reject fonksiyonunu da kullanabilirdik.

var promise = new Promise(function(resolve, reject){

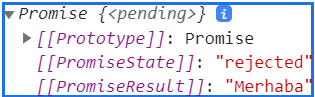
    setTimeout(function(){

        reject("Merhaba");

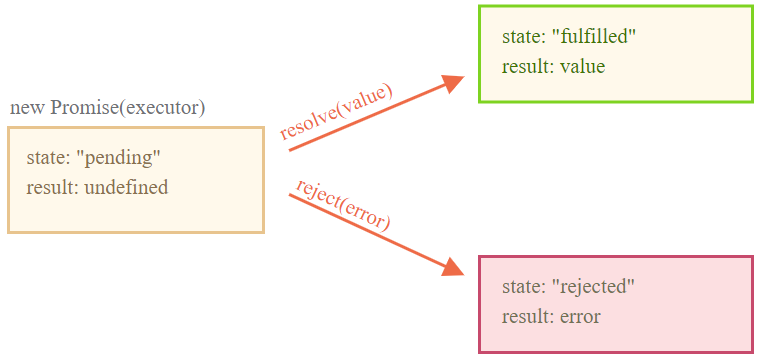
    },2000)

});

console.log(promise);



Bu durumda state özelliği rejected, result özelliği de yine aynı şekilde geriye döndüğümüz değer olacaktır.



function ageControl(age){

    return new Promise(function(resolve, reject){

        if(age >= 18) resolve(true);

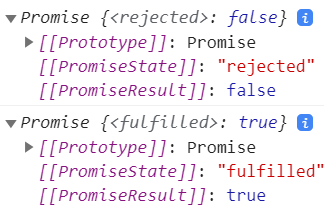
        reject(false);

    });

}

console.log(ageControl(17));

console.log(ageControl(18));



Geriye dönen promise objesinin result’una doğrudan erişemiyoruz. Result’u okumak için then ve catch fonksiyonları kullanılır.

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 1 kez kullanılabilmektedir.

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*

    return new Promise((resolve, reject) => { *//Geriye Promise objesi de döndürebiliyoruz.*

        reject("Test message");

    });

})

.then((response) => {

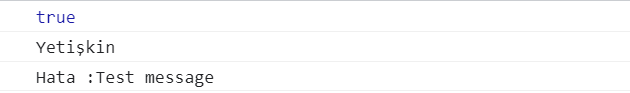
    console.log(response); *//reject döndüğümüz için burası çalışmaz.*

})

.catch((err) => {

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

});



Promise objesi callback fonksiyonlar yerine kullanılabilir.

*//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();

        });

    });

});



Şimdi aynı işlemi Promise’ler ile yapacağız.

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)

    })

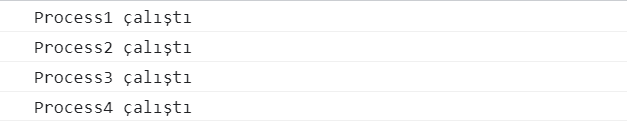
}

function Process4(){

    console.log("Process4 çalıştı");

}

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



Aynı örnek üzerinde **async** ve **await** anahtar kelimelerini kullanabiliriz.

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.

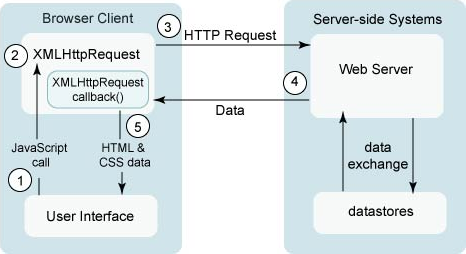
Promise objesi ECMAScript 6 ile birlikte gelmiştir. [Edge+]

async, await anahtar kelimeleri ECMAScript 2017 ile birlikte gelmiştir. [Edge 15+]

**24.Asenkron Programlama**

**AJAX**

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



**‘**Veri alışverişi XML ya da text formatında yapılabilir.

**‘**AJAX işlemleri(GET, POST, PUT, DELETE Request) için XMLHttpRequest objesi kullanılır.

Objenin özellik ve metotlarını konsol penceresinden inceleyebilirsiniz.

var ajax = new XMLHttpRequest();

console.log(ajax);

Örnek bir AJAX uygulaması :

var ajax = new XMLHttpRequest();

ajax.open("GET","./test.txt",true);

ajax.onreadystatechange = function(){

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

        console.log(this.responseText);

    }

}

ajax.send();



Öncelikle bir XMLHttpRequest() objesi oluşturuyoruz.

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

İlk parametre istek türü. Veri çekmek için **GET**, veri eklemek için **POST**, güncelleme işlemi 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. Senkron istek için false değeri verilebilir. Ancak false değerinin kullanılması tavsiye edilmemektedir. Çünkü sunucu tarafındaki işlem zaman alabilmekte ve bu durum browser’ın kitlenmesine sebep olmaktadır. Zaten senkron yapı şuan kaldırılma sürecindedir.

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 özelliği 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 kodları ve anlamları :

202 – OK

404 – Not Found

Http durum kodları ve anlamları : <https://restfulapi.net/http-status-codes/>

Son olarak send() metodu ile isteği gönderiyoruz. Server tarafına veri gönderilirken bu metot send(string) şeklinde kullanılmaktadır.

**‘**status ve readyState mantığı :

var ajax = new XMLHttpRequest();

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

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

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

console.log("-----")

ajax.onreadystatechange = function(){

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

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

}

ajax.send();

*tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu*

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

var ajax = new XMLHttpRequest();

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

ajax.onprogress = function(){

    console.log(this.readyState); *//3*

}

ajax.onload = function(){

    console.log(this.readyState); *//4*

}

ajax.send();

**‘**responseText : **String** türündeki verileri okumak için kullanılır

**‘**responseXML : **XML** verileri okumak için kullanılır.

**‘**abort() : İsteği iptal eder. send() metodundan sonra kullanılmalıdır, önce kullanılırsa bir etkisi olmaz.

**‘**getAllResponseHeaders() : Başlık bilgisinin tamamını döndürür.

**‘**getResponseHeader(string) : İlgili başlık bilgisini döndürür.

**‘**setRequestHeader(header, value) : Gönderilecek verilere başlık bilgisi eklemek için kullanılır.

var ajax = new XMLHttpRequest();

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

ajax.onload = function(){

    if(this.status == 200){

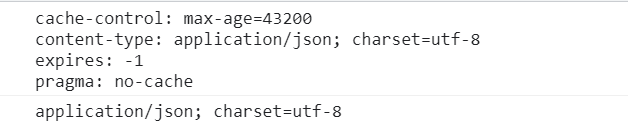
        console.log(this.getAllResponseHeaders());

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

    }

}

ajax.send();



Şimdi AJAX kullanarak fake bir web api ile veri alışverişi yapacağız.

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);*

*// });*

fetch() **Fonksiyonu**

AJAX’a alternatif olarak kullanabileceğimiz bu fonksiyon window objesi içinde bulunmaktadır ve geriye bir Promise objesi döndürmektedir. Tarayıcı desteği [Edge 14+]

fetch() fonksiyonu ile yapılmış bir GET request :

class FETCH{

    get(url){

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

        fetch(url).then((response) => {

            if(response.status == 200){

                return response.json();

            }

            else{

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

                    reject("Hata meydana geldi : " + response.status);

                });

            }

        }).then((response) => {

            return resolve(response);

        }).catch((error) => {

            return reject(error);

        });

    });

    }

}

var obj = new FETCH();

var result = obj.get("https://jsonplaceholder.typicode.com/albums");

result

.then(response => console.log(response))

.catch(error => console.log(error));

**25.JSON**

JavaScript Object Notation(Javascript Obje Notasyonu). **JSON**, Javascript obje sözdizimi ile yazılmış bir metindir(text - string).

**‘**Veri saklamak ve taşımak için kullanılır.

**‘**JSON dosyaları **.json** uzantılıdır.

**‘**JSON metinlerinin **MIME** türü **application/json**’dır.

**‘**AJAX ile sadece string veya xml türünde veriler taşıyabiliyoruz. Zaten JSON da bir string olduğu için JSON formatındaki response’u okurken responseText kullanmaktayız.

**‘**Storage’a sadece string türünde veriler kaydedebiliyoruz. Obje ve diziler storage’a JSON formatında eklenebilir. 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

var 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, number, boolean, array, object, null

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

string kullanmak :

“name”: “John”

number kullanmak :

"age": 30 veya "PI": 3.14

boolean kullanmak:

"sale": true

array kullanmak :

"employees": ["John", "Anna", "Peter"]

object kullanmak :

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

null kullanmak :

"job": null

JSON.parse() **Metodu**

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

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

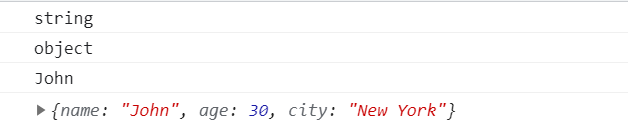
var obj = JSON.parse(json);

console.log(typeof json); *//string*

console.log(typeof obj); *//object*

console.log(obj.name); *//John*

console.log(obj);



JSON.stringify() **Metodu**

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

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

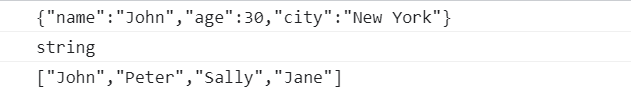
var json = JSON.stringify(obj);

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

console.log(typeof json); *//string*

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

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



**İpuçları**

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

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

var json = JSON.stringify(obj);

console.log(json); *//{"name":"John","today":"2021-08-29T17:08:15.978Z","city":"New York"}*



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

var obj = {name: "John", city: "New York", test\_function: function(){return 1}};

var json = JSON.stringify(obj);

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



Fonksiyonları string'e çevirip daha sonra stringify() kullanın. Alıcıda eval()ile yeniden fonksiyonlara dönüştürebilirsiniz.

var obj = {name: "John", city: "New York", test\_function: function(){return 1}};

obj.test\_function = obj.test\_function.toString();

var json = JSON.stringify(obj);

console.log(json);

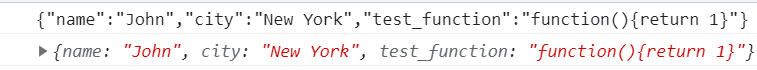
*//{"name":"John","city":"New York","test\_function":"function(){return 1}"}*

var obj2 = JSON.parse(json);

console.log(obj2);

*//{name: "John", city: "New York", test\_function: "function(){return 1}"}*

*//test\_function'u eval ile fonksiyona dönüştürmeniz gerekiyor.*



**26.XML**

XML(Extensible Markup Language – Genişletilebilir İşaretleme Dili), veri saklamak ve taşımak için kullanılan bir dildir.

**‘**XML **W3C** tarafından geliştirilmiştir.

**‘**XML dosyaları **.xml** uzantılıdır.

**‘**XML dosyalarının MIME türü **text/xml** veya **application/xml** olabilir. XML belgesi sıradan kullanıcılar tarafından okunabiliyorsa text/xml, okunamıyorsa application/xml kullanılır.

Örnek bir XML dosyası(test.xml) :

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

<employees>

    <employee>

        <name>Eren</name>

        <age>27</age>

    </employee>

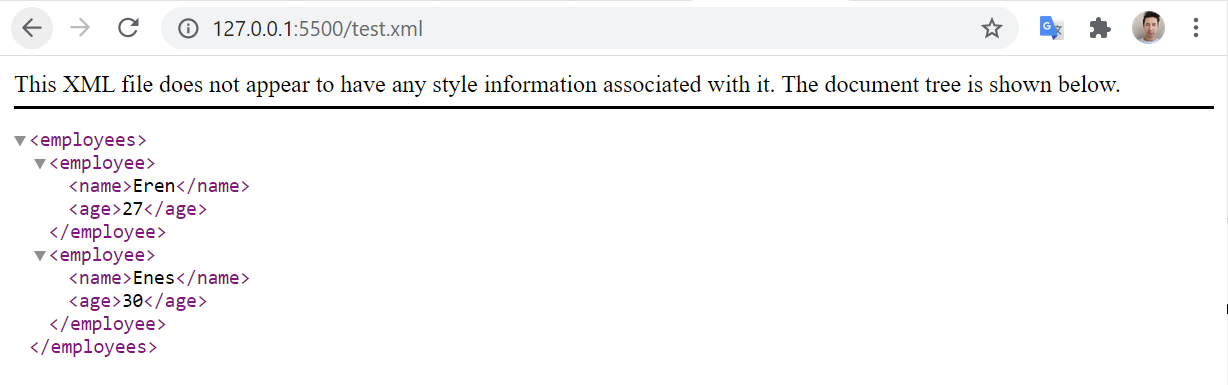
    <employee>

        <name>Enes</name>

        <age>30</age>

    </employee>

</employees>



En dıştaki elemente **root element**, root element içindeki elemenlere de **child element** denmektedir.

Javascript ile **XML DOM** kullanarak XML verileri üzerinde dolaşabilirsiniz. İşte örnek bir uygulama :

function getXMLData(callback){

    var ajax = new XMLHttpRequest();

    ajax.onload = function(){

        if(this.status == 200){

            callback(this.responseXML, null);

        }

        else{

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

        }

    }

    ajax.open("GET", "./test.xml", true);

    ajax.send()

}

getXMLData(function(response, error){

    var response = response; //XML DOM objesi

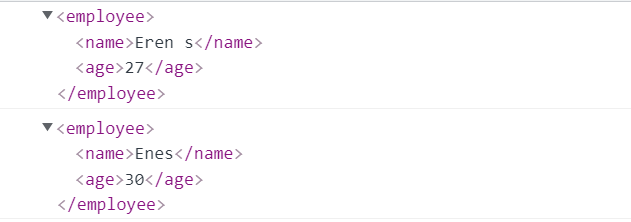
    var elements = response.childNodes[0].children;

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

        console.log(elements[i]);

    }

});



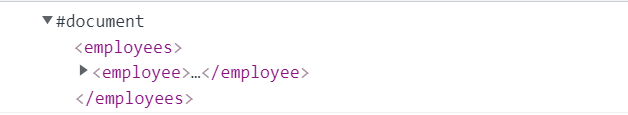
Javascript ile string bir XML veriyi XML DOM nesnesine yükleyerek işlemlerinizi gerçekleştirebilirsiniz.

var data = "<employees><employee><name>Eren</name><age>27</age></employee></employees>";

var parser = new DOMParser();

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

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 veya çift tırnak kullanabilirsiniz.

**‘**XML elementleri isimlendirilirken harf, rakam veya özel karakterler(\_ ve -) kullanılabilir. Ancak XML elementleri rakam ile başlayamazlar.

**‘**XML dosyaları içerisinde açıklama metinlerine yer vermek için <-- --> kullanılır.

**‘**XML verisi içinde bazı özel karakterlerin kullanımı hataya neden olur. Bu özel karakterleri doğrudan kullanmak yerine entity referanlarını kullanabilirsiniz.

< : &lt;

> : &gt;

& : &amp;

“ : &quot;

‘ : &apos;

**27.İpuçları**