JavaScript Notlar

Enes Bayram'ın JavaScript videolarını izlerken aldığım notları sizin de yararlanmanız için paylaşıyorum.

JavaScript sayesinde dinamik yapılar oluştururuz. JavaScript kullanmadan yapılan web siteleri statiktir. Web uygulamaları kullanmak isteyen herkes öğrenmelidir. Auto rename, better comments, javascript console utils, javascript(ES6) code snippets gibi eklentileri VSCode'a indirmemiz JavaScript projeleri yaparken ve JavaScript öğrenirken fazlasıyla yardımcı olacaktır.

JavaScript'i html sayfasına dahil etmek için head veya body tagları içine <script> </script> olarak yazarız. JavaScript'in istediğimizi yapması için html etiketlerinin en altına body'nin içine koymamız daha iyi olur. Console.log("") konsola şunu yaz bunu yaz gibi bir işe yarıyor. App.js'te kodları yazdıktan sonra index'ten ulaşmak istiyorsak index.html içine yine body'nin en alt kısmına <script src="app.js"></> yazarız.

JavaScript'te çıktı vermek için console.log(), document.write() veya alert() kullanırız. Yan yana gelmesini istiyorsak document.write istemiyorsak arada boşluk istiyorsak document.writeln yazarız. Console.clear(); kullanırsak konsolu temizlemiş oluruz.

Let a=5; Let b= 10; console.log(a+b); yazarsak konsolda 15 yazar.

Window objesi JavaScript'nin en geniş objesidir.

Console.log(document.location.host); dediğimizde uygulamayı kurduğumuz host'u konsoldan alabiliriz.

alert("merhaba"); yazarsak sayfamızda merhaba adlı uyarı yazısı çıkar.

Window tüm web sayfasını kapsar document objesi de window objesi içerisinde tanımılanmış bir objedir. Document. Veya window. Diyerek içindeki her şeyi görebiliriz.

console.error(“HATA OLUŞTU") dersek hata oluştuyu konsolda kırmızı bir arkaplanla görürüz. Warn dersek sarı bir şekilde uyarı olarak gelir. Konsol çıktı işlemlerini almak için kullandığımız yerdir. Konsoldan’da kodlarımızı yazabiliriz.

Debugger; kullanarak yazdığımız yüzlerde kod içinden hatanın nerede olduğunu bulacağız, source kısmından. Bazen bulunduğumuz proje üzerinden dışarıdaki sunucuların API adreslerine post-get istekleri yapacağız bunu network üzerinden görebileceğiz. Local Storage’leri yönetebilmek ve görebilmek için application kısmını kullanacağız.

JavaScript’te // veya // kullanarak yorum satırı kullanabiliriz. Komut Ö’ye basarak hepsini yorum satırı haline getirebiliriz. Better Comment eklentisi sayesinde ! ? TODO: * koyduğumuzda yorum satırının rengi değişir */

Scope

scope kapsam anlamına gelir. Global scope,funvtion scop ve block scope olarak 3'e ayrılır. JavaScript'te değişken tanımlamak istiyorsak var let ve const ile tanımlayabiliriz. Global scope en geniş scope'tur hiçbir sınırı engeli yoktur örneğin:

var a= 5; while(a>3){

} if(true){ console.log(a); } function method1(){

}

yazdık a'nın değerini her iki kısımda da alabiliriz. her yerden erişilebilir. global scope'te bir değişken tanımlamak için tanımlanan değişkenin hiçbir kıvırcık parantez içinde olmamalı. en üstte olmalı.

Funciton Scope function içerisinde erişilebilen scope'tur tanımlanan değişken function'ın içinde olmalıdır. Dışarıda görülmez function'dan erişilir.

Block Scope ise function'un içinde for while if do gibi döngüler açarsak bunlar block scope olur. Fonksiyonun dışında bir yerde tanımlarsak bu block scope olur.

VAR LET CONST

Geliştirdiğimiz uygulamada bazı değerleri tutmak isteyeceğiz ve ihtiyacımıza göre kullanmak isteyeceğiz değişkenler bu işe yarar. JavaScrip'te değişken tanımlamak için var let ya da const kullanırız. let sayi=10; şeklinde.

Var ile tanımlanan her şey function scope olur. if else for gibi şeylerin içine tanımladıklarımız da var'la tanımlıyorsak function scope olur. Bunun eksisi ram bellekte çok fazla yer kaplamasıdır.

Let ve Const block scope özelliğne sahiptir. sadece if else vs gibi blokların içinden erişilebilir.

var a=5; var a=10; Yazdık konsolda en son yazdığımız 10 gözükür. Aynısını let ve const'ta kullanamayız. Var'da tek değişken üzerinde birden fazla değer alabilirken let ve const alamaz.

*Let ve Const arasındaki farkta const'un değişmez sabit değiştirilemez olmasıdır. Const'un değeri değiştirilemez. Ler'in değiştirilebilir. COnst'la tanımladığımız şeyleri objelerde de değiştiremeyiz ama objenin içindekileri değiştirebiliriz. Hepsi için geçerli.

string veri tipinde bir değişken tanımlamak istiyorsak tek ya da çift rınak içinde yazmamız gerekir.

let isim= "Buse"; console.log(isim); console.log(typeof isim); //değişkenin tipi

sayı,rakam yazmak istiyorsak tırnak kullanmamamız gerek.

let sayi1= 6; rakamı tırnak içeriinde yazarsak string olarak algılar ve tınrak içindeki iki rakamı toplarsak yan yana yazdırır. int'i int'le string'i string'le toplayabiliriz.

boolen true ve false'tan oluşur.

console.log(5>2); yazarsak konsolde true değeri döner. boolen'da true ve false olmak üzere 2 ihtimal döner.

null boş demektir.

let a=null; console.log(a); dersek konsolda null yani boş döner. Bir değişkeni ilk önce null yaparak sonrasında içine değer atayabiliriz.

undefined

let a; console.log(typeof a); oluşturulmuş ama değişken atanmamış olanlara undefined denir.

oject veri tipi

let insan={ bu object veri tipinde olmuştur. isim:"Buse"; soyisim: "Çetin"; yas: 18; } object veri tipi gelişmiş bir veri tipidir.

primit,ve ilkel refereans tipler modern veri tipleridir.

dizi tanımlamak istiyorsak let rakamlar=[1,2,3];

şeklinde tanımlanır.

ARİTMETİK OPERATÖRLER

= = Atama Operatörü

- = Toplama Operatörü
- = Çıkarma Operatörü
- = Çarpma Operatörü / = Bölme Operatörü % = Mod Alma Operatörü (Kalan Hesaplama- Bir Sayıyı Başka Bir Sayıya Böldüğümzde Kalanı Hesaplamak İçin Kullanılır.) ++ = Bir Arttırma Operatörü -- = Bir Eksiltme Operatörü ** = Üs Alma Operatörü

! İşlem yaparken işlem önceliği için parantezlere dikkat edin!!! ! Türkçe karakter kullanmamaya dikkat edin.

let sayi1=5; let sayi2=10; console.log(sayi1+sayi2); ya da console.log(sayi2/sayi1); console.log(sayi1*sayi2/(sayi1));

let a= 10; a++; console.log(a); yazarsak konsolde 11 yazar.

console.log(5**2); yazarsak konsolda 25 yazar.

ATAMA OPERATÖRLERİ

= = Atama Operatörü. Sağdaki değeri soldaki değere atar. == = Eşittir Operatörü. 5==5 gibi. İçindeki değerler eşitse veri tiplerinin bir önemi yoktur. === = Hem tiplerinin hem de değerlerinin eşit olup olmadığına bakar. Tiplere de baktığı için =='ten faklıdır. += = Sayı değerini arttırır. let sayi=4; sayi+=2; console.log(sayi); dersek konsolda 6 sayısını görürüz

-= = Sayı değerini eksiltir. let sayi=4; sayi-=2; console.log(sayi); dersek konsolda 2 sayısını görürüz. Bölü de aynı şekilde çalışır.

! Alt+Shift+S ile kodlarınızı otomatik toplayabilirsiniz.

*= = Sayı değerinin üssünü alır ve atar. /= = Sayı değerinin bölümünü sayıya atar %= = Kalanı(Modu) sayıya atar. **= = Sayının kuvvetini alıp sayının içine yazdırır.

MANTIKSAL OPERATÖRLER

&& = Ve = Birden çok ihtimal varsa ve hepsinin aynı true ya da false değerini almak istiyorsak kullanırız. || = Veya = Birden çok ihtimal varsa ve sadece birinin koşulu sağlaması yeterli oluyorsa veya operatörünü kullanırız. ! = Değil = Bir değerin zıttını almak için kullanılır.

Tüm bu operatörleri kullanarak bir örnek yapacak olursak kısaca şöyle yapabiliriz: let yas=20; let para=3500; let saglikliMi=false; let hakSayisi=3; console.log((para>3000&& hakSayisi>1) || (yas>18 && saglikliMi))

KARŞILAŞTIRMA OPERATÖRLERİ

== : Eşit Mi ! = : Eşit Değil Mi?

> : Büyük Mü? < : Küçük Mü? = : Büyük Veya Eşit Mi? <= : Küçük Veya Eşit Mi?

console.log(12<=12); yazarsak 12 12'ye küçük olmadığı fakat eşit olduğu için konsolda true değerini alırız.

DİYALOG KUTULARI*

```
-Alert
-Prompt
-Confirm

Bu 3 metotta aslında window objesi içerisinde tanımlanmış 3 metottur. Bir metot window
Kullanıcıya uyarı vermek istiyorsak Alert, kullanıcıdan değer almak istiyorsak Prompt,

alert("F5'e Basmayınız!");

eğer aldığımız isim değerini konsolda yazdırmak istiyorsak aşağıdaki şekilde kullanırı
let isim = propmt("İsminizi Giriniz!");
console.log(isim);
Kullanıcıdan aldığımız değerler her zaman boolen'da olsa number'da olsa string döner.

confirm("Silmek istediğinize emin misiniz?");
```

KOŞUL YAPILARI

if(koşul){ kodlar } else(koşul sağlanmazsa parantez yok süslü parantezler sadece){

}

Örneğin notunuz 50'den büyükse (if) bunu yazdır değilse (else) bu kodları çalıştır. Koşul true olursa if false dönerse else çalışır. İkisi aynı anda çalışmaz. If olmak zorundadır ama Else'e gerek yoktur kullanmak zorunda değilsiniz

*KOŞUL YAPILARI - PRATİK

let not=65; if(not>50){ console.log("Geçtiniz! Tebrikler! Notunuz:" +not); } else{ console.log("Kaldınız. Notunuz: " +not); }

KOŞUL YAPILARI - PRATİK Ortalama Bulma Örneği

```
String değerleri sayısal değerlere dönüştürebilmek için Number metodunu kullanırız.
vize1 %30, vize2 %30, final %40

let vize1= Number(prompt("Vize 1 Notunuzu Giriniz:"));
let vize2= Number(prompt("Vize 2 Notunuzu Giriniz:"));
let final= Number(prompt("Final Notunuzu Giriniz:"));

let ortalama = (vize1*0.3) + (vize1*0.3) + (final*0.4)

if(ortalama>=60){
    alert("Dersten Geçtiniz " + ortalama);
    console.log("Dersten Geçtiniz ");
}
else{
    alert("Kaldınız..." +ortalama);
    console.log("Dersten Kaldınız...");
}
```

IF & Else If & Else Yapısı

```
Birden fazla ihtimal ve koşulun olduğu durumlarda kullanılır. Hepsi birlite çalışmaz.
Yalnızca bir tanesi çalışır.

if(koşul){
}
else if(koşul){
}
else if (koşul){
}
```

```
  else{
  }
```

IF & Else If & Else Yapısı - Yol Ayrımı Uygulaması - 1. Yol /2. Yol /3. Yol

let secim= prompt("Lütfen gitmek istediğiniz rotayı seçiniz.");

if(secim==1){ alert("Seçilen rota"+ "secim"+". rotadır."); } else if(secim==2){ alert("Seçilen rota"+ "secim"+". rotadır."); } else if{ alert("Seçilen rota"+ "secim"+". rotadır."); } else{ alert("Seçilen rota"+ "secim"+". rotadır."); }

Birden Fazla If Kullanmak

Derleyici ilk if'e bakar ilk if sağlıyorsa 2. if'e gidiyor ve son olarak 3. koşula bakıyor. Bunların her biri ayrı bir koşul sağlanmayan koşul çalışmaz.

if(koşul){

} if(koşul){

} if(koşul){

}

Çoklu If Yapısı

-Kullanıcının adını ve tckimlik numarasını isteyeceğiz kimlik numarası 11 karakterden oluşsun.

let ad= prompt("İsminizi Giriniz :") let teckn= prompt("TC'nizi Giriniz :") console.log(isim);

funciton kontrol(ad, tckn){ if(ad!=""){ if(tckn.length==11){ console.log("İsim ve TCKN girildi.");

```
      }
      else{
          console.log("Lütfen geçerli bir kimlik numarası giriniz.");

      }

  }else{
      console.log("İsim Alanını Boş Bırakmayınız!");
  }
```

}

Kod çalıştırıldıktan sonra metodu sonlandırmak istiyorsak Return anahtar kelimesini kullanabiliriz.

Örnek-2

kontrol2(ad,tckn);

function kontrol2(ad,tckn){ if(ad==""){ console.log("Lütfen isminizi giriniz!"); return; } if(tckn.length!=11){ console.log("Lütfen geçerli bir kimlik numarası giriniz!"); return; } console.log("Başarıyla giriş yapıldı.") }

Beden Kitle Endeksi Bulma

Kiloyu boyun karesine bölerek beden kitle endeksini bulabiliriz.

let boy= Number(prompt("Boyunuzu Giriniz :")); let kilo= Number(prompt("Kilonuzu Giriniz :")); let sonuc= kilo/(boy*2);

if (sonuc<18.5){

console.log("İdeal Kilonun Altındasınız");

} else if(sonuc>=18.5 && sonuc<=24.9){

console.log("İdeal Kilodasınız");

}else if(sonuc>=25 && sonuc<=29.9){

console.log("İdeal Kilonun Üstündesiniz");

}else if(sonuc>=30 && sonuc<=39.9){

console.log("Obezsiniz");

}else if(sonuc>=40){

```
   console.log("Morbiz Obezsiniz");
```

}

BENZİN İSTASYONU UYGULAMASI

1-Dizel : 24.53 2-Benzin : 22.25 3-LPG : 11.1

Gelen müşteriden alacağımız bilgiler: 1-Yakıt Tipi 2- Yüklenecek Yakıt Litresi

let dizel= 24.53 , benzin= 22.25, lpg= 11.1; const yeniSatir= "/r/n"

const yakitMetni = "1-Dizel"+yeniSatir

- "2-Benzin"+ yeniSatir +"3-LPG" +yeniSatir
- "Yakıt Türünüzü Seçiniz: ";

/r/n kullanırsak alt alta yazdırır.

let yakitTipi = prompt(yakitMetni); yakit tipi 1-2-3 değilse kodlar çalışmayacak aşağıdaki kod bloğu bu işe yarıyor. if(yakitTipi=="1"|| yakitTipi== "2"|| yakitTipi==3){ if (yakitMetni=="1"){

```
matematiksel işlem yapacaksak Number'a çevirmemiz gerek.
let yakitLitresi = Number(prompt("Yakıt Litresini Giriniz :"));
```

let bakiye= Number(propmt("Lütfen Bakiyenizi Giriniz: ")); let odenecekTutar= dizel*yakitLitresi; if (odenecekTutar> bakiye){ alert("Bakiyeniz Yetersiz, Fakirsiniz.") } else{ alert("Bakiyeniz Yeterlidir.") } }else if(yakitTipi=="2"){ let odenecekTutar= benzin*yakitLitresi; if (odenecekTutar> bakiye){ alert("Bakiyeniz Yetersiz, Fakirsiniz.") } else{ alert("Bakiyeniz Yeterlidir.") } }else if(yakitTipi=="3"){ let odenecekTutar= lpg*yakitLitresi; if (odenecekTutar> bakiye){ alert("Bakiyeniz Yetersiz, Fakirsiniz.") } else{ alert("Bakiyeniz Yeterlidir.") } }

}else { alert("Lütfen geçerli bir yakıt türü seçiniz!"); }

Switch Case

switch(deger){ case 1: kodlar break; case 2: kodlar break; case 3: kodlar break; default: kodlar break; }

Switch Case Örnek Haftanın Günleri

If-Else gibi caselerdan yalnızca biri çalışır. Break koymazsak diğer caselerda çalışır kodumuzda komplike hatalar olur. Bir case'e girdikten sonra diğerine girmesini istemiyorsak break koymalıyız.

let deger= Number((prompt("Seçiminizi Giriniz :")));

switch(deger){ case 1: alert("Pazartesi") break; case 2: alert("Salı") break; case 3: alert("Çarşamba") break; case 4: alert("Perşembe") break; case 5: alert("Cuma") break; case 6: alert("Cumartesi") break; case 7: alert("Pazar") break;

default: alert("1 İle 7 Arasında Geçerli Bir Değer Giriniz") break;

}

Switch Case ATM Örneği

1- Bakiye Görüntüleme 2- Para Çekme 3- Para Yatırma 4- Çıkış

let bakiye=1000; let yeni= "/r/n"

let metin= "1-Bakiye Görüntüleme"+yeni +"2-Para Çekme"+yeni +"3-Para Yatırma"+yeni +"4-Çıkış"+yeni +"Lütfen Bir Değer Seçiniz!"

let secim=prompt(metin); switch(secim){ case "1": alert("Bakiyeniz :"+bakiye); break; case "2": let cekilecekTutar=Number(prompt("Çekmek İstediğiniz Tutarı Giriniz :")); if(cekilecekTutar<bakiye){ bakiye= bakiye- cekilecekTutar; alert("Kalan Bakiye :"+bakiye); }else{ alert("O kadar zengin değilsiniz"+ yeni+ "Bakiyeniz :" +bakiye +" "+ "Çekilecek Tutar: "+ cekilecekTutar); } break;

```
      case "3":
          let yatirilacakTutar= Number(prompt("Yatırılacak Tutarı Giriniz: "));
          bakiye= bakiye+ yatirilacakTutar;
          alert("Güncel Bakiyeniz :"+ bakiye);
      break;

      case "4":
          alert("Sistemden Başarıyla Çıkış Yapılmıştır...");
      break;

      default:
          alert("Lütfen 1 – 4 Arasında Değer Giriniz !!!");
          break;


}
```

Switch Case -Son Örnek- TYT Puan Hesaplama

1- Türkçe 40 -4 puan 2-Matematik 40 3- Sosyal Bilimler 20 4- Fen Bilimleri 20

-------> 100 Puanı ÖSYM Veriyor. -------> Okul Puanı Max 60 Veriyor.

Başlangıç değeri olarak hepsine 0 veriyoruz.

let turkcedogru, turkceyanlis =0; let matematikdogru, matematikyanlis =0; let sosyaldogru, sosyalyanlis =0; let fendogru, fenyanlis =0; let puan= 0; let okulpuani;

let yeni= "/r/n"; let mesaj= "TYT Puan Hesaplamasına Hoşgeldiniz..."+yeni +"1- Puan Hesapla"+yeni +"2- Çıkış Yap";

let secim= propmt(mesaj);

switch(secim){ case "1":

```
      okulpuani= Number(prompt("Okul Puanınızı Giriniz :"))

      turkcedogru= Number(prompt("Türkçe Doğru Sayısı :"));
      turkceyanlis= Number(prompt("Türkçe Yanlış Sayısı :"));

      matematikdogru= Number(prompt("Matematik Doğru Sayısı :"));
      matematikyanlis= Number(prompt("Matematik Yanlış Sayısı :"));

      sosyaldogru= Number(prompt("Sosyal Doğru Sayısı :"));
      sosyalyanlis= Number(prompt("Sosyal Yanlış Sayısı :"));

      fendogru= Number(prompt("Fen Doğru Sayısı :"));
      fenyanlis= Number(prompt("Fen Yanlış Sayısı :"));

      ! 4 yanlış bir doğruyu götürüyor !
      let dogruNet= turkcedogru+matematikdogru+sosyaldogru+fendogru;
      let YanlisNet= turkceyanlis+matematikyanlis+sosyalyanlis+fenyanlis;
      let kalandogru= dogrunet- (YanlisNet/4);
      puan= (kalandogru*4)+ 100+ okulpuani ;
      alert("TYT Puanınız: "+puan);

      break;

  case "2":
      alert("Uygulamadan Başarılı Bir Şekilde Çıkış Yaptınız.");
      break;

      default:
          alert("Geçerli Bir Değer Giriniz");
          break;
```

Tür Dönüşümleri

strings,numbers,boolens,undefined and null. == Primitive object,function == Modern

Örneğin bir veri tabanımız var ve veritabanından değer almak istiyoruz değer number olarak geldi ama biz string'e dönüştürmek istiyorsak tür dönüşümlerini kullanmamız gerekir. Bir tipten bir tipe dönüştürmeye denilir.

let a =5; let b= "10";

console.log(a+b); yazarsak 510 değeri gelir. Türler aynı olmadığı için toplanamaz.

! String Veri Tipinden Number Veri Tipine Dönüştürme

let c= Number(b); console.log(typeof c); console.log(a+c); dersek 15 cevabını alırız.

ya da

let a =5; let b= Number("10"); console.log(a+b); şeklinde yaparsak da 15 cevabını alırız.

veya

parseInt() kullanırız. parseInt window objesi içinde tanımlanmış bir metottur. parseInt yerine parseFloat da kullanılabilir. parseInt ve parseFloat arasındaki fark parseInt kullanırsak virgüllü sayının noktadan sonrasını atıp tam sayı olarak verir. parseFloat ise virgülüyle birlikte alır.

let a =8; let b= parseInt("12"); console.log(a+b); şeklinde kullanılır.

}

! Number Veri Tipinden String Veri Tipine Çevirme

let x= 55; Örneğin bunu string veri tipine çevirmek istiyoruz. let x= String(55); olarak yazmamız gerekir. ya da: let x= (55).toString(); yazarak kullanabiliriz.

! Boolen Tipindeki Değeri String'e Çevirme

let sonuc= String(true); console.log(typeof sonuc); console.log(sonuc);

! Sayı Olmayan Bir Şeyi Number'a Çevirmeye Çalışırsak NaN (Not a Number) Adlı Bir Hata Alırız.

! Bir Objeyi Ya Da Array'i String'e Çevirme

let rakamlar = String([1,2,3,4]);

! Proje Yaparken Veya Hata Alınca Breakpoint(Debugger) Kullanmayı Unutmayın. Ayrıca Tooltip'leri Okumanızda Kodu Anlamanız Konusunda Size Hayli Yardımcı Olacaktır. ! Tooltip Açıklama Satırı Gibidir. Yazılımcıyı Bilgilendiren Küçük Yorum Satırlarıdır Diyebiliriz. ! Breakpoint (Debugger)' Bir Hata Aldığımızda Veya Bir Kod Bloğunu Anlamadığımızda Kodları Adım Adım Çalıştırmak İçin Kullanırız. Bunu Kullanabilmek İçin debuger; Denilen Bir ! Anahtar Kelime Kullanmamız Gerek. Adım Adım Çalıştırmak İstediğimiz Projede NErden BAşlamak İStiyorsak O Kod Blogu Öncesi debugger; Yazarız. Debugger'ı Nereye Koyarsak Orada ! Çalışmaya Başlar.

Döngülere Giriş For/While/Do-While/ForEach*

Bir şeylerin sürekli tekrarlanmasını istediğimiz yerlerde kullanırız. Tek satırda yazdığımız kodu istediğimiz kadar çalıştıran kod parçalarıdır. ForEach dizilerle birlikte kullanılır o yüzden şimdilik ForEach'i üstünkörü göreceğiz.

For Döngüsü Çalışma Yapısı

```
for(degişken; koşul; arttırma-azaltma){
    kodlar
}

Bu Kodları Çalıştırdıktan Sonra Alttaki Kodları Çalıştırıyor.

for(let i=1;i<10;i++){
    console.log(i);
}
```

For Döngüsü Örnek

```
1'den 10!a Kadar Yazdıralım
```

for (let i=1;i<=10;i++){ console.log(i) }

For Döngüsü Örnek - 2

```
1'den 10'a Kadar Olan Çift Sayıları Yazdıralım

+=2 i'nin üzerine 2 koy her seferinde 2 arttır demektir.
```

for (let i=0; i<=10;i+=2){ console.log(i); }

```
Tekleri Yazdıramk İsteseydik i'yi 1'den Başlatırdık:
```

for (let i=1; i<=10;i+=2){ console.log(i); }

İ Çift Sayı Olunca Buse Tek Sayı Olunca Berat Yazdıran Kod:

for(let i=0;i<=10;i++){ if(i%2==0){ console.log("Buse"); } else{ console.log("Berat"); } }

50'den 1'e Kadar Olan Sayıların Toplamı:

let toplam=0; for(let i=1;i<=50;i++) { toplam+=i; console.log(i); } console.log("Toplam :" ,toplam); For Döngüsünün İçine Değil Dışına Yazmamızın Sebebi Her Döngü Döndükten Sonra Toplamı Görmek Yerine Döngü Bitince Toplamı Görmek İstememiz.

Break Ve Contiune Anahtar Kelimeleri *

```
1'den 10'a Kadar Olan Sayıları Yazdıralım 8'e Geldiğimizde Döngüden Çıkalım.
```

for (int i=1;i<=10;i++) { if(i==8){ break; } }

ya da

let sayac=1;

while(sayac<=10) { console.log(sayac); if(sayac==8) { break; } sayac++; }

Bu Gibi Döngülerde Break Kullanılır. Yani Break'i Dilediğimiz Zaman Döngüden Çıkmak İçin Kullanırız. Contiune İse Döngüyü 1 Kere Kırmak İçin Kulanılır. Örneğin Konsolda 1-10 Arası Sayıları Yazdırmak İstiyoruz Fakat 8'in Yazmasını İstemiyorsak Contiune Kullanırız:

while(sayac<=10) { sayac++; if(sayac==8) { contiune; } console.log(sayac); }

Çarpım Tablosu Uygulaması

1*1=1 1*2=2 1*3=3 gibi gibi gidecek

for (let i=1;i<=10;i++){ for (let j=1;j<=10;j++){ console.log(i+"x"+j+"="+(i*j)); } console.log(">>>>>>>>>>>>>"); }

Dıştaki Döngü 1 Kere İçteki Döngü 10 Kere Dönüyor. 1'i 10'a Kadar Çarptıktan Sonra Koşul Sağlamadığı İçin Çizgiye Çekti Ve Bir Daha Döngünün İçine Girerek 2'den Devam Etti Ve Aynı Şekilde 10'a Kadar Devam Edip Durdu.

Asal Sayı Bulma Uygulaması

Asal Sayı: 1'e Ve Kendisinden Başka Böleni Olmayan Sayıdır. Math Sınıfının İÇinde Floor Diye Bir Sınıf Var Bu Sınıf Bizim Sayımızı En Yakın Int Değerine Yuvarlar. 7.7 Veya 7.9 Yazdığımızda Sayıyı Direkt 7'ye Çekiyor.

```
let sayi= Number(prompt("Sayı Giriniz : "));
let sonuc= true;

    for(let i=2;i<= Math.floor(sayi/2);i++){
        if(sayi%i==0){
            Asal Değildir
            sonuc= false;
            break;
        }
```

```
    }
    if(sonuc){
        alert(sayi + "asaldır.");
    }
    else{
        alert(sayi + "asal değildir.");
    }
```

Faktöriyel Hesaplama

```
Faktöriyel hesaplama bir sayının kendinden 1'e kadar olan çarpımına denir.

let sayi= Number(propmt("Bir Sayı Giriniz :"));
let carpim=1;

for(let i=1;i<=sayi;i++){
    carpim=carpim*i;
}
alert ("Sonuç : "+carpim);

 Armstrong Sayısı Uygulaması
```

153,407,370 sayıları armstrong sayılardır. Armstrong sayı bir sayının her bir rakamın kökünün toplamının kendisini vermesidir.

```
let sayi= prompt("Bir Sayı Giriniz :");
for(let i=0;i<sayi.length;i++){
    let rakam = sayi.charAt(i);
    toplam+=rakam*rakam*rakam;
}

if(Number(sayi)==toplam){
    alert("Armstrong Sayısıdır.");
}
else{
    alert("Armstrong Sayısı Değildir.");
}


!Length Sayının Basamağı Demektir i<sayi.length dediğimizde sayının basamağı kadar dön
! Number'a çevirirsek sayının içinde char length gibi özellikleri göremeyiz bu yüzden
!  yer alan özelliklerdir. Bu yüzden bunları Number'a çevirirsek bu özellikleri kullan
```

Metotlar (Methods)

Parça paarça kodlayıp bu parçaları tek bir yerde birleştiriyoruz. Kodların okunmasının tek satırda yazarak kolayca kullabiliriz.

Parametresiz Ve Geriye Değer Döndürmeyen Metot Tanımlamak

Bir metot oluşturuken function yazıp sonrasında metotun ismini veriyoruz sonrasında
Bu metotu çalıştırmak için yazdir(); şeklinde metotu çağırıyoruz.

```
function yazdir(){
  console.log("Buse Nur");
}

yazdir();
```

Bu metotu bir daha çağırmak istiyorsak bir kere daha yazdir(); yazıyoruz. Metotla iş

Örnek 2

```
function topla(){
  console.log(5+7);
}

topla();
```

Parametreli Metot Tanımlama

```
function yazdir(isim,soyisim){
  console.log(isim+" "+soyisim);
}
yazdir("Buse Nur", "Çetin");
```

Şeklinde Tanımlanan Metotlar Parametreli Metotlardır. Sonrasında yazdir ("Berat", "Ç
Buse Nur Çetin yazdırdıktan sonra Berat Çetin yazdıracaktı.

Bir Sayının Küpünü Alan Bir Metot Yazalım

```
    cube(3);
    cube(5);

    function cube(sayi){
     console.log(sayi*sayi*sayi);
    }
```

Kullanıcıdan Yaşını Alalım 18'ten Küçükse Ehliyet Alamasın

Metotlar 2 kelimeden oluşuyorsa ikinci kelimenin ilk harfi büyük yazılır.

```
let yas = Number(prompt("Yaşınızı Giriniz :"));
kontrolEt(yas);

function kontrolEt(yas){
    if(yas>18){
        console.log("Ehliyet Alabilirsiniz.");
    }
    else{
        console.log("Ehliyet Alamazsınız.");
    }
}
```

Bir yerde parantez aç kapa varsa bu metottur. kontolet(); gibi.

Geriye Değer Döndüren Metot Tanımlamak

Örneğin Bir Sayının Küpünü Alalım Ve Sonucu Geriye Döndürelim.

```
cube(3);

function cube(sayi){
   let sonuc = sayi*sayi*sayi;
   console.log(sonuc);
}
```

Burdan çıkan sonucu alıp başka bir yerde kullanmak isteyebiliriz. Bu şekilde yazarsak kıvırcık parantezler içinde kalır bu sonuca başka bir yerden erişemeyiz. Başka bir y anahtar kelimesini kullanırız.

```
let donenDeger = cube(3);
console.log(donenDeger);

function cube(sayi){
   let sonuc = sayi*sayi*sayi;
   return sonuc;
}
```

Sonucu metotun çağrıldığı yer olan cube(3)'e geri döndüreceği için dönen değeri yakal

```
kareAl(donenDeger);

function kareAl(sayi){
   let sonuc = sayi*sayi;
   console.log(sonuc);
}

! Return Anahtar Kelimesinden Sonra Yazılan Hiçbir Kod Çalışmaz.
! Return Bir Değeri Metotun Dışarısına Çıkarmak, Taşımak İçin Kullanılır.
! Void Geriye Değer Döndürmeyen Metot Demektir.
```

Harf Sayısı Uygulaması

```
Örneğin kullanıcı A harfi cümlenin içinde kaç defa geçiyor diye soracak bizde kullanı
Şimdilik Örnek Olarak Bir Metin Yazıp Kullanıcıya Hangi Harfi İstediğini Soralım:

let metin = "Şuanda Bursa'da JavaScript Öğreniyorum."
let harf(prompt("Harfi Giriniz :"));

funcion bul(harf){
   for(let i=0;i<metin.length;i++){
       if(metin.charAt(i)==harf){
           toplam +=i;
       }
   }
   return toplam;
}

Büyük Harf/ Küçük Harf Hassasiyetinin Olmamasını İstiyorsak:

funcion bul(harf){
   for(let i=0;i<metin.length;i++){
       if(metin.charAt(i).toLowerCase==harf.toLowerCase()){
           toplam +=i;
       }
   }
   return toplam;
}

Kullanırız. toLowerCase büyük olan harfi küçük harfe dönüştürerek hassasiyeti önler.
```

Mükemmel Sayı Uygulaması

```
6-28-496 Mükemmel Sayılardır. Bir Sayının Tam Bölen Sayısının Toplamı, Sayının İki Ka
```

```
6 = 1,2,3,6 Toplamları= 12  6*2=12

perfectNumber(6);
perfectNumber(496);
perfectNumber(407);
perfectNumber(2);

function perfectNumber(number){
   let toplam=0;
   for(i=2;number/2;i++){
       if(number%i==0){
           toplam+=i;
       }
   }
   toplam+= 1+number;
   if(toplam== number*2){
       console.log("Mükemmel Sayıdır.");
   }
   else{
       console.log("Mükemmel Sayı Değildir.");
   }
}
```

Decimal To Binary Conversion (10'luk Sayıyı 2'liğe Çevirme Uygulaması)

```
Decimal ondalıklı sayı demektir. Binary ikilik sayı demektir 0 ve 1'lerden oluşur.

convertDecimalToBinary ondalıklı sayıyı ikilik sisteme çevirmemize yarayan bir metott
Math.floor metotu sayesinde sayıyı ikiye böldükten sonra noktadan sonrasını atıp tam

Örneğin 10 sayısını kullanalım. Siz İstediğiniz Sayıyı Kullanabilirsiniz.

convertDecimalToBinary(10);

function convertDecimalToBinary(number){

   let binary= "";
   while(true){
       binary+=(number%2).toString();
       number=Math.floor(number/2);
       if(number==1){
           Artık Bölmek Yok Döngüden Çıkacağız.
           binary+=1;
           break;
       }
   }
   let result = reverse(binary);
```

```
    console.log("Sonuç :"+ result);
}

Bu Metotu Ters Çevirmemiz Gerekiyor. Ters Çevirmek İçin:

function reverse(binary){
    let reverseBinary = "";
    for(let i=binary.length-1;i>=0;i--){
        reverseBinary+=binary.charAt(i);
    }

    return reverseBinary;
}
```

Binary To Decimal Conversion (İkilik Sayıdan Onluk Sayıya Çevirme)

```
İkilik Sayıyı Onluk Sayıya Çevirmek İstiyorsak Sağdan Sola Doğru 2 üzeri 0, 2 üzeri 1
  Sonrasında Bu Sayıları Bunlara Denk Gelen Sayılarla Çarpıp Hepsini Toplarız.

let binary= "1001"
let us=0;

function convertBinaryToDecimal(binary){
    let toplam =0;
    for(let i=binary.length-1;i>=10;i--){
        toplam+=  Number(binary.charAt(i)) * Math.pow(2,us);
        us++;
    }
    console.log("Sonuç :"+toplam);
}
convertBinaryToDecimal(binary);

     Metotlar Bitiş
```

Dizilere (Array) Giriş

```
let dizi_ismi= ["Buse", "Berat", "Damla", "Hakan","Okan"];
let dizi_ismi= [1, true, "Buse", '?', null,5.12];
Normalde başka programlama dillerinde farklı türdeki dizileri aynı dizi içinde tanıml
Dizilerde index dediğimiz bir kavram kullanılır. Diziler saymaya 0'dan başlar.
```

Dizilere Giriş

```
! Kutu Parantezi ALT GR + 8 ve 9'u Kullanarak Yapabilirsiniz.

let sayilar= [0,1,2,3,4,5,6,7,8,9];
console.log(sayilar[6]); yazarsak 6 sayısını bize getirir.

Eğer Olmayan Bir Index Değeri İsterseniz Yani 5 Değerli Bir Index'ten 10. Değeri İste
 Diğer Programlama Dillerinde ArrayIndexOutOfBoundException Hatası Alırken JavaScript
 Onun Yerine Undefined Hatası Alırsınız.

let sayilar= [0,1,"Buse",3,4,5,"Nur",7,8,9];
Örneğin Ben Bu Dizideki Buse Değerini Çetin Olarak Değiştirmek İstiyorum Bunu Şu Şeki

sayilar[2]= "Çetin";
console.log(sayilar[2]);

Ya Da

sayilar[sayilar.length-6]="Çetin"

Olarak Kullanabilirsiniz. Indexlerle Erişiyoruz Indexler Üzerinden Değerlerini Değişt

let isimler=["Buse", "Nur", "Çetin", "Berat"];
console.log(isimler);

Olarak Yazdırırsanız Index Numarasıyla Birlikte İsimleri Konsolda Görebilirsiniz.

let karisikDizi= [1, "Buse", 2,3,7,0, true,null,undefiened];
console.log(karisikDizi[3]);

İki Şekilde Dizi Oluşturabiliriz

1) let dizi1=[];
2) ket dizi2= new Array(); Bu Şekilde Dizi Tanımlamak Nesne Tanımlayarak Dizi Tanımla

! Diziler Aslında Object Veri Tipindedir.

console.log(typeof dizi2); Yazarsak Bunu Konsolda Görebiliriz.
```

FOREACH DÖNGÜSÜ

Önceki Notlarda For,While,Do While Döngülerini Yazmıştım. Bunların Bir Arkadaşı Daha Foreach Döngüsü Sadece Dizilerde Kullanılır. Dizimiz Varsa Ve Dizinin İçindeki Eleman Falan Da Kullanabiliriz Fakat Foreach Kullanım Kolaylığı Açısından Daha İyidir.

Foreach Kullanımı

```
let isimler =["Buse","Nur","Çetin","Berat"];
isimler.forEach(function(isim)){
   console.log(isim);
}

Şeklinde Foreach Döngüsü Kullanılabilir. Bunu For Döngüsüyle De Yazabiliriz:

let isimler =["Buse","Nur","Çetin","Berat"];
for (let i=0;i<isimler.length;i++){
   console.log(isimler[i]);
}

While İle De Yapılabilir:

let sayac=0;
while(sayac<isimler.length){
   console.log(isimler[sayac]);
   sayac++;   Sayacı Arttırmazsak Sonsuza Kadar Döner Bu Yüzden Arttırmayı Unutmayın.
}

!!! Diziler 0'dan Başlar 1'den Başlarsanız İlk Değeri Kaybedersiniz.
```

Dizi Metotları

Push= Dizinin Sonuna Eleman Ekler, Ayrıca Dizinin Uzunluğunu Döner.

Unshift= Dizinin Başına Eleman Ekler, Eleman Sayısını Geriye Döner.

Pop= Dizinin Sonundan Eleman Siler.Eleman Sayını Döner.

Shift= Dizinin Başından Eleman Siler. Eleman Sayını Döner.

Splice(Index,Incdex)= Eleman Eklemek Ve Silmek İçin Kullanılır.

toString= Diziyi Stringe Çevirir.

Join= Diziyi Stringe Çevirir. Farkı İse Araya Eleman Ekleyebilmemizdir.

Concat= Dizileri Birleştirmek İçin Kullanılır.

Slice(Dilimlemek)= Diziyi İstenilen Yerden Bölüp Yeni Dizi Oluşturur.

Length= Dizinin Uzunluğunu Verir.

Reverse= Orjinal Dizinin Elamanlarını Ters Çevirir.

Split(Bölmek)= Belirli Bir İfadeye Bölüp Diziye Çevirmek.

indexOf= Dizideki Elemanın İndex Numarasını Verir.

Includes= Dizi Verilen Elemanı İçeriyor Mu Ona Bakar.

```
! Ayrıca Sadece Tooltipleri Okuyarak Tüm Bu Dizilerin Ne İşe Yaradıklarını Görebilirs
```

Push Metot

```
let arabalar= ["Ford","Mercedes","TOGG","Toyota"];
console.log(arabalar.length)
let diziUzunluk= arabalar.push("Şahin");
console.log(diziUzunluk);
```

Unshift Metot

```
arabalar.unshift("Toros");
console.log(arabalar);
```

Pop Metot

```
let silinen=arabalar.pop();
console.log(arabalar);
```

```
console.log(silinen);
```

Shift Metot

```
arabalar.shift();
console.log(arabalar);
```

Splice Metot

```
Splice İle De Hem Ekleyip Hem Sileriz. Aradaki Fark Splice İle Sadece Başa Sona Değil

console.log(arabalar)
arabalar.splice(0,0,"Tofaş");
console.log(arabalar);

arabalar.splice(1,2);
console.log(arabalar);

arabalar.splice(2,2,"Porsche");
console.log(arabalar);
```

toString Metot

```
console.log(typeof arabalar);
let stringArabalar=arabalar.toString();
console.log(typeof arabalar);
console.log(stringArabalar);
```

Join Metot

```
let stringArabalar=arabalar.join(".");
console.log(stringArabalar);
```

Concat Metot

```
let arabalar1=["TOGG","Mercedes","BMW"];
let arabalar2=["Tofaş","Şahin","Toyota"];
```

```
let birles= arabalar1.concat(arabalar2);
console.log(birles);
```

Slice(Dilimlemek) Metot

```
let arabalar=["TOGG","Mercedes","BMW","Tofaş","Şahin","Toyota"];

console.log(arabalar);
let ayri=arabalar.slice(2);
console.log(ayri);
```

Length Özelliği (Metot Değil Özellik)

```
console.log(arabalar.length);
```

Reverse Metot

```
console.log(arabalar);
arabalar.reverse();
console.log(arabalar);
```

Split Metot

```
let isimler=["Buse","Nur","Berat","Çetin"];

let isimlerDizi= isimler.split(",");
console.log(typeof isimlerDizi);
```

indexOf Metot

```
let index= arabalar.indexOf("bmw1");
if (index==0){
   console.log("Belirtilen Eleman Vardır.");
}else{
```

```
    console.log("Belirtilen Eleman Yoktur");
}
```

Includes Metot

```
let sonuc = arabalar.includes("BMW");
console.log(sonuc);
```

Örnek

```
Elimizde birkaç tane ürün olsun kullanıcıdan prompt metotuyla hangi ürünü istediğini
kaç tane varsa sayısını konsola yazdıralım.

let urun1={
   isim: "ACER Bilgisayar",
   kategori: "Teknoloji",
   fiyat= 12.342
}
let urun2={
   isim: "Macbook Bilgisayar",
   kategori: "Teknoloji",
   fiyat= 23.431
}
let urun3={
   isim: "Lenova Bilgisayar",
   kategori: "Teknoloji",
   fiyat= 12.321
}
let urun4={
   isim: "Excalibur Bilgisayar",
   kategori: "Teknoloji",
   fiyat= 23.423
}

let urunler=[urun1,urun2,urun3,urun4];

let KullaniciUrunIsmı = prompt("Bir Ürün İsmi Giriniz...");

FiltreliUrunlerDoldur(urunler);
FiltreliUrunleriYazdir(urunler);

function FiltreliUrunlerDoldur(urunler){
    urunler.forEach(function(urun){
   if(urun.isim.toUpperCase().includes(KullaniciUrunIsmı.toUpperCase(),0)){
       filtreliUrunler.push(urun);
   }
```

```
});
}

function FiltreliUrunleriYazdir(urunler){
   urunler.forEach(function(urun){
       console.log("-------------------------");
       console.log("|"+urun.isim+"|"+urun.fiyat+"|"+urun.kategori);
       console.log("-------------------------");
   });
}
```

Diziler Örnek

```
Ürün Bulma Uygulaması

let urun1 = {
   isim: "Macbook",
   kategori: "Teknoloji",
   fiyat: 83.239;
}
let urun2 = {
   isim: "Lenova",
   kategori: "Teknoloji",
   fiyat: 43.239;
}
let urun3 = {
   isim: "Ipad",
   kategori: "Teknoloji",
   fiyat: 23.239;
}
let urun4 = {
   isim: "Huwaei",
   kategori: "Teknoloji",
   fiyat: 53.239;
}
let urun5 = {
   isim: "Casper",
   kategori: "Teknoloji",
   fiyat: 13.239;
}
let urun6 = {
   isim: "Excalibur",
   kategori: "Teknoloji",
   fiyat: 82.339;
}

Gerçek hayatta böyle kullanılmaz. Veri tabanından çekilir.

let urunler = [urun1,urun2,urun3,urun4,urun5,urun6];
```

```
let filtreli =[];
let kullanıcıIsım= propmt("Bir Ürün İsmi Giriniz.");

FiltreliUrunleriDoldur(urunler);
FiltreliUrunleriYazdir(filtreli);

function FiltreliUrunleriDoldur(urunler){
   urunler.forEach(function(urun){
   if(urun.isim.toUpperCase().include(kullanıcıIsım.toUpperCase,0)){
       filtreli.push(urun);
   }
});
}

function write FiltreliUrunleriYazdir(urunler){
   urunler.forEach(function(urun){
       console.log("-----------------");
       cosole.log("|"+ urun.isim+"|"+urun.fiyat+"|"+urun.kategori);
       console.log("-----------------");
   });
}
```

Örnek 2 - Kitap Bulma Uygulaması

```
Raflar oluşturacağız ve yazdığımız kitapların kaçıncı rafta olduğunu kullanıcı soruncaya

let kitap1 ={
   isim= "Her Şeyi Düşünme",
   yazar: "Anne Bogel",
   fiyat: 25
   raf: "1.5.RAF"
}
let kitap2 ={
   isim= "Hiçbir Karşılaşma Tesadüf Değildir",
   yazar: "Hakan Mengüç",
   fiyat: 32
   raf: "2.3.RAF"
}
let kitap3 ={
   isim= "İnsan Neyle Yaşar",
   yazar: "Tolstoy",
   fiyat: 34
   raf: "3.3.RAF"
}
let kitap4 ={
   isim= "Zafer Sızlanarak Kazanılmaz",
   yazar: "Haluk Tatar",
   fiyat: 45
   raf: "4.5.RAF"
```

```
}
let kitap5 ={
   isim= "Şeker Portakalı",
   yazar: "Jose",
   fiyat: 50
   raf: "1.2.RAF"
}


let kitaplar [kitap1,kitap2,kitap3,kitap4,kitap5];

5 raftan oluşan 25 kitap tutabilen bir sistem gibi düşünün.

let raf11={kod: "1.1.RAF", goster:false}
let raf12={kod: "1.2.RAF", goster:false}
let raf13={kod: "1.3.RAF", goster:false}
let raf14={kod: "1.4.RAF", goster:false}
let raf15={kod: "1.5.RAF", goster:false}

let raf21={kod: "2.1.RAF", goster:false}
let raf22={kod: "2.2.RAF", goster:false}
let raf23={kod: "2.3.RAF", goster:false}
let raf24={kod: "2.4.RAF", goster:false}
let raf25={kod: "2.5.RAF", goster:false}

let raf31={kod: "3.1.RAF", goster:false}
let raf32={kod: "3.2.RAF", goster:false}
let raf33={kod: "3.3.RAF", goster:false}
let raf34={kod: "3.4.RAF", goster:false}
let raf35={kod: "3.5.RAF", goster:false}

let raf41={kod: "4.1.RAF", goster:false}
let raf42={kod: "4.2.RAF", goster:false}
let raf43={kod: "4.3.RAF", goster:false}
let raf44={kod: "4.4.RAF", goster:false}
let raf45={kod: "4.5.RAF", goster:false}

let raf51={kod: "5.1.RAF", goster:false}
let raf52={kod: "5.2.RAF", goster:false}
let raf53={kod: "5.3.RAF", goster:false}
let raf54={kod: "5.4.RAF", goster:false}
let raf55={kod: "5.5.RAF", goster:false}

let raflar= [
[raf51,raf52,raf53,raf54,raf55]
[raf41,raf42,raf43,raf44,raf45]
[raf31,raf32,raf33,raf34,raf35]
[raf21,raf22,raf23,raf24,raf25]
[raf11,raf12,raf13,raf14,raf15]

];

function rafolustur(){
```

```
      console.clear();
      let satir="";
      for (let i=0; i<raflar.length;i++){
          for (let j=0; j<5;j++){
              satir+="|"+(raflar[i][j].goster ? raflar[i][j].kod: "-----")+"";

          }
          console.log(satir);
          console.log("---------------");
          satir="";
      }
   }

   rafolustur();

   Rafları Oluşturduk Şimdi Kullanıcı Ekrandan Kitap Girince Kitabı Bulduralım.

   function kodBul(kitapIsmi){
      let rafKod= null;
      kitaplar.forEach(function(kitap){
          if(kitap.isim.toUpperCase().includes(kitapIsmi.toUpperCase(),0))
          {rafKod=kitap.raf;
      }
      });
      return rafKod;
  }

  function raftaGoster(rafKodu){
      for(let i=0;i<raflar.length;i++){
          for (let j=0, j<5; j++){
              if (raflar[i][j].kod==rafKodu){
                  raflar[i][j].goster=true;
                  break;
              }
          }
      }
  }

  rafOlustur();
  let kitapIsim= prompt("Lütfen Bir Kitap İsmi Giriniz!");
  let rafKod= bul(kitapIsmi);
  console.log(rafKod);
```

if (rafKod!=null){ raftaGoster(rafKod); raftaOlustur();

```
  }else{
    alert("Girdiğiniz kitap kütüphanemizde bulunmamaktadır.")
  }
```

String Sınıfının Metotları

```
* charAt()
* concat()
* indexOf()
* lastindexof()
* toUpperCase()
* toLowerCase()
* trim()
* slice()
* substring()
* replace()
* split()
* valueOf()
* startsWith()
* endsWith()

charAt() Kullanımı:
let kurs ="JavaScript Öğren"
let index= kurs.charAt(4);  Yazıdaki 4. karakteri yani harf/rakamı alır.
console.log(karakter);

concat Kullanımı:
String Birleştirmek İçin Kullanılır.
let sonuc = kurs.concat(tarih);
console.log(sonuc);

indexOf Kullanımı:
charAt() indexi verip karakteri buluyorken indexof karakteri alıp indexi bulur.
let index = kurs.indexOf("J");
console.log(index);

lastIndexOf Kullanımı:
Vermiş Olduğumuz Kelimenin Index'ini Yakalamamızı Sağlar.
let index = kurs.lastIndexOf("Kursu");
console.log(index);

toUpperCase Kullanımı:
Tüm karakterleri büyük harfe dönüştürür. Orjinalini bozmuyor,kopyasını alıp büyük har
let sonuc = kurs.toUpperCase();
console.log(sonuc);

toLowerCase Kullanımı:
Tüm karakterleri küçük harfe dönüştürür.
let sonuc = kurs.toLowerCase();
console.log(sonuc);

Trim Kullanımı:
Örneğin kullanıcıdan kullanıcı adı aldık, bu metot kullanıcı, adını boşluklu gönderse
console.log(kurs.trim());
```

```
Slice Kullanımı:
Bu metotlar dilimlemek için kullanılıyor.
console.log(kurs);
console.log(kurs.slice(7,10)); 7. Index'ten Başla 10'a Kadar Al, 10 Dahil Değil.

Substring Kullanımı:
Slice gibidir farkı ise eksi değerleri alabilmesidir.
console.log(kurs);
console.log(kurs.substring(0,6));

Replace Kullanımı:
Örneğin JavaScript Öğren Yerine JavaScript Kursu Yazmak İstiyoruz.
console.log(kurs);
console.log(kurs.replace("Öğren","Kursu"));
Değiştirmek İstediğiniz Kelime Ve Neyle Değiştirmek İstediğinizi Yukarıdaki Gibi Yazı

Split Kullanımı:
Dizide İstedğimiz Bir Bölümden İstediğimiz Bölüme Kadar Olan Kısmı Ayırıp Dİzi Şeklin
console.log(kurs);
let dizi= kurs.split(" ");
console.log(dizi);

valueOf Kullanımı
console.log(kurs.valueOf());

startsWith Kullanımı:
Bir Değer Verip O Değerle Başlayıp Başlamadığını Kontrol Edebiliriz.
console.log(kurs);
console.log(kurs.startsWith("M"));  M İle Başlamadığı İçin False Dönecek

endsWith Kullanımı:
Sonu Şununla Mı Bitiyor. Her İkisinde De Harf De Kelime De Verebilir.
console.log(kurs);
console.log(kurs.endsWith("Modern"));

    Math (Matematik) Objesi Metotları

* Floor
* Ceil
* Round
* Max
* Min
* Random
* Abs
* Sqrt
* Pow
-----> PI Sayısı

Math Sınıfı Da Aslında Window Objesi İçinde Tanımlanmış Bir Objedir.

1 - Floor
Noktadan sonrasını atarak tam sayı değeri verir. Yani 3.99 yazsanız bile çıktı 3 olur
```

```
let a =3.15
console.log(Math.floor(a));

2 - Ceil
Floor'a bir benzer metot da ceil metotu. Noktadan sonrasına bakmaksızın sayıyı bir üs
console.log(Math.ceil(a));

3 - Round
Round ise en yakın olan değere yuvarlar. 3.11 yazarsanız 3'e, 3.99 yazarsanız 4'e yuv
console.log(Math.round(a));

4 - Max
Number tipinde bir dizi alır, geriye yine number olarak döner.
console.log(Math.max(1,2,3,4,50));
Buradaki en büyük değer olan 50'yi döner.

5 - Min
Number tipinde bir dizi alır, geriye yine number olarak döner.
console.log(Math.min(1,2,3,4,50));
Buradaki en büyük değer olan 1'i döner.

6 - Abs
Bir sayının mutlak değerini almak için kullanılır.
let b = -12;
console.log(Math.abs(b));

7 - Sqrt
Bir sayının karekökünü almak için kullanılır.
console.log(Math.sqrt(169));

8 - Pow
Bir sayının üssünü almak için kullanılır.
console.log(Math.pow(2,4)); şeklinde kullanılır 4 tane ikinin çarpımını sonuç olarak

9 - PI Sayısı
console.log(Math.PI);
PI bir metot değil bir property(özellik)tir.

10 - Random
Random metotu rastgele değerler üretmek için kullandığımız bir metottur. Çok sık kull
console.log(Math.random());
Rastgele sayılar oluşturduktan sonra noktadan sonrasından kurtulmak için aşağıdaki gi
console.log(Math.floor(Math.random()*100));
Kod en dıştan içe doğru okunduğu için bu şekilde yazarız.

Bunu parçalı olarak yazmak istersek şu şekilde yazabiliriz:

let randomDeger = Math.random();
let sonuc = randomDeger*100;
let yuvarla = Math.floor(sonuc);

console.log(randomDeger);
console.log(sonuc);
```

```
console.log(yuvarla);
```

Date (Tarih) Metotları

Date tarih işlemleri yaptığımız bir objedir ve içince birçok metot vardır. Bu da wind
Tarih nasıl oluşturulur sorusu ile başlayalım. Yeni bir date objesi oluşturmak için n

Bir tarih oluşturuyorsunuz ve get diyerek ayını gününü dakikasını saniyesini alabiliy

console.log(typeof tarih); yazarsak bunun bir obje olduğunu görürüz.

tarih.toString(); derseniz bunun tipini string'e değiştirebilirsiniz.

Tarihi ya let tarih = new Date(); şeklinde ya da yılını ayını gününü saatini dakikası

```
let tarih = new Date();
console.log(tarih);
```

Yazarsanız şuanda tarih neyse onu görürsünüz.

Tarih objesinin içerisine girdiğinizde get ve set metotlarını görürsünüz.

Get Metotları:

console.log(tarih.getTime()); yazarsanız da zamanı number tipinde konsola görebilirsi

console.log(tarih.getFullYear()); yazarsanız size içinde bulunduğunuz yılı döner.

console.log(tarih.getDate()); yazarsanız günü konsol da görebilirsiniz. Bu metot tari

* console.log(tarih.getDay()); yazarsanız da haftanın gününü alır yani Pazartesi günü

* console.log(tarih.getHours()); yazarsanız saati alırsınız.

* console.log(tarih.getMilliseconds()); yazarsanız milisaniyeyi gösterir.

* console.log(tarih.getMinutes()); yazarak da dakikayı alabilirsiniz.

* Yeniden hatırlatmak istiyorum. Bunların her birinin birer metot olduğunu unutmayın.

* Bunları bir veri güncellendi ne zaman güncellendi, kişi sisteme kayıt oldu ne zaman

* console.log(tarih.getMonth()); yazarsanız içinde bulunduğunuz ayı verir fakat aylar
* console.log(tarih.getMonth()+1); yazarak gerçek ayı alabilirsiniz.

* console.log(tarih.getSeconds()); yazarak saniyeyi alabilirsiniz.

* console.log(tarih.toLocalDateString()); yazarsanız tarihi gün ay yıl şeklinde alabi

* console.log(tarih.toLocalTimeString()); yazarsanız tarihi saat dakika saniye şeklin

* console.log(tarih.toLocalString()); yazarsanız da tarih(gün/ay/yıl) ve saat(saat/da

* Set Metotları:

* Set metotlarını yeni değerler vermek için kullanırız. Örneğin tarihi aldınız fakat

* tarih.setData(24);
* console.log(tarih);

* Yazarsak tarihin gününü 24 olarak değiştirmiş oluruz.

* tarih.setHours(15); yazarsak saati değiştirmiş ve 15 olarak göstermiş oluruz.

* tarih.setMonth(11); yazarsak ayı değiştirmiş oluruz.

* tarih.setMinutes(50); yazarsak dakikayı değiştirmiş oluruz.

* tarih.setHours(tarih.getHours()+2); yazarsak şuanki saatin üzerine 2 saat eklemiş

    * Değer Ve Referans Tipleri

Bir değer bir de referans tipi adında 2 tane değişkenimiz var. Bir değişkeni oluşturd
Değer tipinde dediğimiz değişkenler aslında ilkel (primitive) veri tipleridir. String

let isim ="Buse Nur";
console.log(typeof isim); yazarsak bunun string veri tipinde yani ilkel (primitive) b

Referans tipleri ise ilkel olmayan tiplerdir. Object, array, function referans tipind

let dizi = [1,5,8,3]; yazarsak bu ilkel değil referans türündedir

Peki değer ve referans tipi arasındaki fark nedir?

Ram bellekte S ve H adında iki değişken vardır. Değişken değerlerimiz ram belleğin st
Stack bölümünde değer tipler, hip bölümünde referans tipler tutulur.

Değer Tipleri Örnek:

```
let a = 7;
let b = a;

console.log("a :" + a);
console.log("b :" + b);

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

b=10;
console.log("a :"+ a);
console.log("b :"+ b);
```

Gördüğünüz gibi b'de değişiklik yapmamıza rağmen bu değişiklik a'yı etkilemiyor. Değe

Referans Tipleri Örnek:

```
let dizi1 = [1,2,3];
```

```
let dizi2 = [1,2,3];

if(dizi1==dizi2){
   console.log("Eşittir.");
} else{
   console.log("Eşit Değildir.")
}

//console.log(dizi1);
//console.log(dizi2);
```

Bu şekilde yazarsak konsolda çıkan çıktı "Eşit değildir." olarak çıkacaktır.
Bunların eşit olmasına rağmen eşit değildir çıkmasının sebebi, referans tipinde olduk
Burda yazdığımız kontrol sistemine göre aslında değerlerinin değil referans adresleri

```
let dizi1 = [1,2,3];
let dizi2 = [dizi1];

if(dizi1==dizi2){
   console.log("Eşittir.");
} else{
   console.log("Eşit Değildir.")
}
```

Şeklinde yazarsak dizi2 dizi1'in adresine bakacağı için konsolda "Eşittir."'i görebil

```
let dizi1 = [1,2,3];
let dizi2 = [dizi1];

dizi2.push(12); dizi2'ye 12'yi ekliyoruz.

console.log(dizi1) yazdırdığımızda değişikliği dizi2'de yapmış olmamıza rağmen 12 diz
Genel olarak değer ve referans tipi arasındaki temel fark budur.
```

* DOM (Document Object Model) Nedir?

DOM HTML dökümanları içerisinde bulunan nesnelere kolaylıkla erişim sağlamak ve üzeri
Dinamik olarak o HTML sayfasında değişiklik yapmamızı sağlar. W3 Schools Document Mod

* Document Objesi

Document objesi Window objesi içinde tanımlanmış bir objedir. Çok büyük bir objedir i
Bir matruşka örneği verelim. Widnow objesi en büyük matruşka, document onun bir küçüğ

```
console.log(window); yazarsanız Document objesinin Window içinde olduğunu görebilirsi

Location Özelliği:

let value;
console.log(document.location)
value= document.location.href;  yazarsanız hangi adres üzerinde çalıştığınızı konsola
value= document.location.hostname;  yazarsanız hangi localhostta çalıştığınızı göster
```

```
value= document.location.port;  yazarsanız hangi port üzerinde çalıştığınızı konsolda
value= document.location.pathname;  yazarsanız hangi klasörün içinde ne isimli dosyad
console.log(value);
```

Charset Özelliği

```
let value;
console.log(document)
value= document.characterSet; yazarsanız sayfanın hangi dil koduyla çalıştığını konso
console.log(value);
```

Title Özelliği

```
let value;
console.log(document)
value= document.title; yazarsanız üzerinde çalıştığınız projenin başlığını konsolda g
console.log(value);
```

Link Ve İtem Özelliği

```
let value;
console.log(document)
value= document.links;  yazarsanız sayfadaki a etiketlerini alabilirsiniz.
value= document.links[4];  yazarsanız 4. indexteki link konsolunuzda görebilirsiniz.
value= document.links.item(4).id;  yazarak da 4. indexteki linkin ID'sini konsola yaz
value= document.links.item(4).getAttribute("id");  yazarak da ID'yi konsolda görebili
value= document.links.item(4).getAttribute("class");  yazarak da 4. indexteki linkin
value= document.links.item(4).classList;  yazarak da elementin kaç tane class'a sahip
value= document.links.item(4).classList[3];  yazarak da class listesinin 3. indexinde
value= document.forms;  yazarak sayfadaki formları konsolda görebiliriz.
value= document.forms.item(1);  yazarak 1. indexteki formu konsola yazdırabiliriz.
value= document.forms.item(1).id;  yazarak index numarası 1 olan formun ID'sini konso

console.log(value);
```

Domain Özelliği

```
let value;
console.log(document)
value= document.domain; yazarsanız host adresinizi görebilirsiniz.
console.log(value);
```

ContentType Özelliği

```
let value;
console.log(document)
value= document.contentType; yazarsanız sayfanın içeriğinin tipini görebilirsiniz. Te
console.log(value);
```

* Document objesi kullanarak oluşturduğum basit todo list örneğine bakmayı ve aynısın
bu örnek için oluşturulmuştur. İlerleyen zamanlarda çalışan, çok daha iyi gözüken, di

```
    * Selectors (Seçiciler) - Style Özellikleri

JavaScript'te 3 şekilde seçebiliyoruz. Birisi ID birisi classname birisi tag name.

Aynı class ismini birçok yerde tanımlayabilirsiniz ama ID'ler unique olmalıdır. Aynı

getElementById : ID'ye göre elementi yakalar.
getElementByClassName: Class ismine göre yakalar.
getElementByTagName: Etiket ismine göre yakalar.


getElementsById Örnekler:

let value;
value= document.getElementsById("todoAddButton"); Bunun anlamı dökümanımın içinde ID'
console.log(value);

const button = document.getElementsById("todoAddButton");
console.log(button);
console.log(button.id);  Yazarak da butonun ID'sini konsola yazdırabiliriz.

Bir elementin içindeki class type name attribute olarak geçer.

const button = document.getElementsById("todoAddButton");
console.log(button);
console.log(button.getAttribute());  Yazarak todoAddButton ID'sine sahip elementin cl

const button = document.getElementsById("todoAddButton");
console.log(button);
console.log(button.getAttribute("id"));  Yazarak da elementin ID'sini alabiliriz.

const button = document.getElementsById("todoAddButton");
console.log(button);
console.log(button.className);  Yazarak butonun class ismini konsola yazdırabiliriz.

const button = document.getElementsById("todoAddButton");
console.log(button);
console.log(button.getAttribute("class"));  Yazarak butonun class ismini yine konsola

const button = document.getElementsById("todoAddButton");
console.log(button);
const classListesi = button.classList;
console.log(classListesi);  Yazarak seçtiğimiz elemanın classlarını liste şeklinde ko

const button = document.getElementsById("todoAddButton");
console.log(button);
const classListesi = button.classList[3];  Yazarak 3. indexteki class'ı konsolda göre
console.log(classListesi);

const button = document.getElementsById("todoAddButton");
console.log(button);
```

```
const classListesi = button.classList;
classListesi.forEach(function(className){
   console.log(className);
}  Yazarak classlar üzerinde dönebiliriz.
console.log(classListesi);

let buttonText= button.textContent;  Yazarak yakalamış olduğumuz elementin içindeki t
let buttonText2= button.innerHTML;  textContent ile aynı işi yapar.
console.log(buttonText);
console.log(buttonText2);
```

textContent ve innerHTML arasındaki fark şudur:

button.textContent= " Todo Ekleyin " Yazdırırsanız b etiketlerini textContent
button.innnerHTML= " Todo Ekleyin " Yazdırırsanız b etiketini olması gerektiğ
Yani kısaca innerHTML, HTMl etiketlerini algılayabiliyorken textContent algılayamaz.

getElementsByClassName Örnekler:

```
const todolist= document.getElementsByClassName("list-group-item");  Class ismi list-
console.log(todolist);  Bu class ismine sahip tüm elemanları bunu kullanarak konsolda

const todolist= Array.from(document.getElementsByClassName("list-group-item"));
todolist.forEach(function(todo){
   console.log(todo);  Yazarak da bu class ismine sahip her bir elementin üzerinde dö
})
console.log(todolist);

const todolist= Array.from(document.getElementsByClassName("list-group-item"));
todolist.forEach(function(todo){
   console.log(todo.className);  Yazarak bu class ismine sahip her bir elementin clas
})
```

getElementsByTagName Örnekler:

```
const forms = document.getElementsByTagName("form");  Yazarak form tagına sahip tüm e
console.log(forms);

const forms = document.getElementsByTagName("form");  Yazarak 0. indexteki forma eriş
console.log(forms[0]);

const forms = document.getElementsByTagName("form");  Yazarak 0. indexteki formun ID'
console.log(forms[0].id);

const forms= Array.from(document.getElementsByTagName("form"));
forms.forEach(function(form){
   console.log(form);   Yazarak her bir formun üzerinde tek tek dönebiliriz. Array ku
})

const todolist= document.getElementByTagName("li");  Yazarak li etiketine sahip eleme
console.log(todolist);
```

```
Şuana kadar getElementById -  getElementByClassName - getElementByTagName metotlarını
Bu metotlar querySelector ve querySelectorAll metotlarıdır.

     * querySelector - querySelectorAll

const clearButton= document.querySelector(#todobutton);  Yazdığımızda ID'si todobutto
console.log(clearButton);

const todoList= document.querySelector(".list-group");  Bu seferde class ismi list-gr
console.log(todoList);

const todoList= document.querySelector(".list-group-item");  Şeklinde yazarsak bu cla
Aynı elemandan 1'den fazla varsa querySelector değil querySelectorAll kullanırız. Cla
console.log(todoList);

const todoList= document.querySelectorAll(".list-group-item");  list-group-item class
console.log(todoList);

const todoList= document.querySelectorAll(".list-group-item")[2]; 2. indexe sahip lis
console.log(todoList);

const todoList= document.querySelectorAll(".list-group-item")[document.querySelectorA
console.log(todoList);

const todoList= document.querySelectorAll("li:first-child");  Yazarak li'lerin ilk ço
console.log(todoList);

const todoList= document.querySelectorAll("li:last-child");  Yazarak li'lerin son çoc
console.log(todoList);

const todoList= document.querySelectorAll("li:nth-child(2)");  Yazarak li'lerin ikinc
console.log(todoList);

const todoList= document.querySelectorAll("li:nth-child(odd)");  Yazarsanız tekleri s
console.log(todoList);

const todoList= document.querySelectorAll("li:nth-child(even)");  Yazarsanız çiftleri
console.log(todoList);

const todoList= Array.from(document.querySelectorAll("li:nth-child(even)"));
todoList.forEach(function(todo){
  todo.style.backgroundColor= "lightgrey";  Yazarak çift sayıya sahip tüm li elementl
})

Özet olarak anlatmak gerekirse JavaScript'te elementleri 3 şekilde seçebiliriz. Ya ID
Fakat bunlardan çok bunların hepsini yapabilen querySelector ve querySelectorAll meto
Class ile seçim yapmak için "." etiket ile seçim yapmak içinse direkt yazarak seçimi
birden fazla yerde kullanılmış ise tüm elementerli seçebilmek için querySelectorAll k

    * Style Özelliği Ve Kullanımı

const todo= document.querySelectorAll(.list-group-item)[0];
const todoList = document.querySelector(.list-group);
```

```
const Button = document.querySelector(#Button);

todo.style.color= "red"; 0. indexteki list-group-item classına sahip elementin yazısı
todo.style.color= "#fff"; 0. indexteki list-group-item classına sahip elementin yazıs

Bunları ezberlemenize gerek yok her bir HTML elementinin style diye bir objesi var ve
W3 Schools gibi sitelerden bu özelliklerin tamamına bakabileceğiniz için diğerleri gi

todo.style.backgroundColor= "pink";  Yazarsak seçili elementin arkaplan rengi pembe o

todo.style.fontWeight= "bold";  Yazarak seçili elementin yazısını bold yani kalın yap

İki kelimeden oluşsan CSS özelliklerinde ikinci kelime her zaman büyük yazılır.

todo.style.paddingTop="5rem";  Yazarsanız seçili elemente üstten 5rem boşluk bırakırs

todoList.style.marginTop= "10px"; Yazarsanız seçili elemente üstten 10px boşluk bırak

todoList.style.marginLeft= "20px" Şeklinde yazarsanız da soldan 20px boşluk bırakabil

Button.style.backgroundColor= "green";  Yazarak butona yeşil arkaplan rengi verebilir

Button.style.fontWeight= "bold";  Yazarak buton içindeki yazıları bold(kalın) yapabil

Button.style.padding="10px";  Yazarak butona 4 bir yandan 10px'lik boşluk vererek dah

Button.style.borderRadius="25px";  Yazarak butonun kenarlarına 25px'lik bir yuvarlakl

Bu özellikleri kullanarak dinamik olarak sayfamızda değişiklikler yapabiliriz. Bunlar
yapabileceğiniz herhangi bir değişikliği style özelliği kullanarak da yapabilirsiniz.
W3 Schools'un websitesine göz atarak tüm özellikleri görebilirsiniz.

    * HTML Elementleri Üzerinde Gezinmek

HTML'in bir anne olduğunu varsayalım bu annenin head ve body adında iki çocuğu vardır
h1, p, img ve body'nin içindeki diğer tüm elementler kardeştir. Aynı şekilde title ve
anneler, bir de çocuklar vardır. Bu örneği daha iyi anlayabilmek için document-todoli

Bu Örnekleri daha iyi anlamak için ToDo projesine bakarak okuyunuz.

const todo = document.querySelector(".list-group-item");
const todoList = document.querySelector(".list-group");

conts card = document.gelElementByClassName("card"); böyle ya da;
const card = document.querySelector(".card"); böyle card elementini seçebilirsiniz
console.log(card); value kullanırsanız her seferinde console log tanımlamak zorunda k

const row = document.querySelector(".row");

let value;

 * Anneden Çocuklara Erişmek *
```

```
value= todoList;
value= todoList.children;

Sadecde 0. indexteki çocuğu yakalamak istiyorsak:

value= todoList;
value= todoList.children[0];

Sadece 1. indexteki çocuğu yakalayalım:

value= todoList.children[1];

Son çocuğu yakalamak için şu kod bloğu kullanılabilir:

value= todoList.children[todoList.children.length-1];

Son çocuğu yakalayıp içindeki yazısını da değiştirebiliriz.

value= todoList.children[3].textContent = "Değişti";

Tüm ToDo elemanlarının üzerinde dönmek istiyorsak:

value = Array.from(todoList.children); yazarız.

value.forEach(function(todo){
   console.log(todo.textContent);
})  yazarak da her bir textContent ögesi üzerinde dönebiliriz.

value=todo;
console.log(value);

* Çocuktan Anneye Erişmek *

value = todo;
value = todo.parentElement; yazarak todo'nun annesine erişilebilir.
value = todo.parentElement.parentElement;  yazarak todonun annesinin de annesine eriş
value = todo.parentElement.parentElement.parentElement;  yazarak da onun da annesine
value = value.parentElement;  yazarak da yukarıdakilerin annesine ulaşılabilir.

Her birini ayrı ayrı da yazdırabilirsiniz:

value todo;

let ul = todo.parentElement;
let cardBody = ul.parentElement; gibi.
let cardElement = cardBody.parentElement;
let row = cardElement.parentElement;
console.log(cardBody);

  * Kardeşler Arasında Gezinmek *

value = todo;
value = todo.nextElementSibling;  Sibling kardeş demek next element sibling ise bir s
```

```
value = todo.nextElementSibling.nextElementSibling;  dersek onun üzerinden de bir son
value = todo.nextElementSibling.nextElementSibling.nextElementSibling;  yaparsak başk
console.log(value);

     * Son Çocuğu Yakalamak İçin *

const todoLastChild = document.querySelector(".list-group-item:nth-child(4)"); yazarı
value = todoLastChild;
value = todoLastChild.previousElementSibling; yazarsak son çocuktan bir önceki elemen
console.log(value);

 * Belirli Bir Elementi Yakalamak *

Todo List yazısını değiştirmeye çalışacağız. Row'u yukarıda tanımladık.

value= row.children[0].children[3].children[0].textContent ="Başlık Değişti";

Pratik yapmak isterseniz seçtiğiniz bir yazıyı değiştirmeye çalışın. Yukarıdan aşağıy

Şimdi de Todo 3 yazan yerin yazısını ve arkaplan rengini değiştirelim:

let todo3 = row.children[0].children[3].children[2].children[2];
todo3.textContent="Değişti.";
todo3.style.backgroundColor="gray";

console.log(value);

* Dinamik Olarak Element Oluşturmak *

Bu kısmı da daha iyi anlamak için todolistproje.html'e bakmayı unutmayın.

  const cardBody = document.querySelectorAll(".card-body")[1];
  const todoList = document.querySelector(".list-group");

  Document sınıfımızın objesiiçerisinde createElement adında bir metot var. Bunu kull

  const link = document.createElement("a");

  Bu elemente ID vermek istersek:

  linkID="blogWebsite"; şeklinde verebiliriz.

  Bu elemente class vermek istersek:

  link.className="btn btn-dark":

  Bu elemente href yani yönlendirme vermek istersek:

  link.href="https://buse.com";

  Bu elemente target vermek istersek de:

  link.target="_blank":
```

Bu elementin içine yazı yazmak istersek:

link.innerHTML="Buton"; yazabiliriz.

console.log(link); yazarak da elementi konsolda tüm özellikleriyle görebiliriz.

Bu elementi sayfaya yerleştirmek için ilk önce yukarıda card-body sınıfına sahip el
querySelectorAll yazmamızın sebebi card-body'den 2 tane olması bu kısmı anlamadıysa

Bu card-body'nin sonuna ekleme yapmak istiyoruz bu yüzden.

cardBody.appendChild(link); yazarak ekliyoruz. Append Child sonuna ilave et, bu çoc

Bunu da yazdıktan sonra sayfada a elementini görebiliyoruz.

Başka bir örnek daha yapalım:

Bu örnekte hem li elementini hem li elementinin içindeki a etiketini hem de i etike

```
const todo = document.createElemet("li");
const todoLink = document.createElement("a");
const i = document.createElement("i");

todoClassName="list-group-item";
todo.innerHTML = "TODO 5";
todoLink.href="#";
todoLink.className= "delete-item";
i.className= "fa fa-remove";
```

a etiketinin içerisine i'yi koymak için:

todoLink.appendChild(i); yazıyoruz.

a etiketini li etiketinin içerisine koymak içinse:

todo.appendChild(todoLink); yazıyoruz.

Şimdi bunların hepsi hazır fakta bu li'yi de ul'nin içine eklememiz gerekiyor. Yuka

todoList.appendChild(todo);

yazıp kaydettiğimizde başarılı bir şekilde sayfamızda görebiliriz.

* Element Silmek *

Bir element nasıl yakalanıp silinir bunu öğreneceğiz. 2 tane yolu var ikisini de göre

Örneklerle öğreneceğiz Todo 1'i nasıl yakalayıp sileriz bakalım:

1. Yöntem:

```
const todoList = document.querySelector(".list-group");
```

```
const todos = document.querySelectorAll(".list-group-item");

todos[0].remove();
todos[1].remove();
console.log(todos);

yazarak 0. ve 1. indexteki todo'yu silebilirsiniz.

Sondaki çocuğu sileceksek:

const todoList = document.querySelector(".list-group");
const todos = document.querySelectorAll(".list-group-item");

todos[todos.length-1].remove(); yaparak da sondaki çocuğu silebiliriz.

Ya da:
const todoList = document.querySelector(".list-group");
const todos = document.querySelectorAll(".list-group-item");
const todo1 = document.querySelector(".list-group-item");

todo1.remove(); yaparak da silebiliriz.

2. Yöntem:

Sileceğimiz elementi annesi(ebeveyni) üzerinden de silebiliriz.

console.log(todos) yaparsanız todo'ların hepsinin NodeList şeklinde olduğunu konsolda

const todoList = document.querySelector(".list-group");
const todos = document.querySelectorAll(".list-group-item");
const todo1 = document.querySelector(".list-group-item");

todoList.removeChild(todos[0]); yazarsanız Todo listesindeki 0. indexteki çocuğu sil

Ya da:

let todoq = todos[0];
todoList.removeChild(todo1); yazarsanız da aynı işlemi yapmış olursunuz.

todoList.removeChild(todos[todos.length-1]); yazarsanız da son çocuğu silebilirsiniz.

todoList.removeChild(todoList.lastElementChild);  yazarak da sonuncu çocuğu yani son

   Elementlerin Yerini Değiştirmek

replaceChild metodunu kullanarak elementlerin yerini değiştirebiliriz. Yine bir örnek

const cardBody = document.querySelectorAll(".card-body")[1]; Card Body'i yakaladık ye

const newTitle = document.createElement("h2"); bir H2 elementi oluşturduk.
newTitle.className="card-title";
newTitle.textContent = "Todo Listesi Yeni";
```

İlk önce yeni etiketimizi sonrasında değiştirmek istediğimiz etiketi yazıyoruz ama es

console.log(cardBody.childNodes); yazarak cardBody'nin node tiplerini görebiliriz.

console.log(cardBody.childNodes[1]); yazarak 1. indexteki node'u yazdırıyoruz.

cardBody.replaceChild(newTitle,cardBody.childNodes[1]); yazarak da yeni etiketimizle

2. parametreyi yani değiştirilmek istenen elementi her zaman Node tipinde ister buna

Events (Olaylar) Nedir?

Türkçe anlamı olaydır. Bir şey tetiklendiğinde meydana gelmesini istediğimiz olayları
Örneğin bir butona tıklamai butona 2 kez tıklama ya da butonun üzerine gelme, üzerind
Events etiketlerine web3 schools'tan göz atabilirsiniz fakat hepsine burda da değinec

Events (Olaylar) Nasıl Kullanılır?

Bu kısımdaki örnekleri anlamak için document-todolistproje.html'e bakarak ilerlemeyi

Örneğin:

<a href="#" id="todoClearButton" onclick="alert('Merhaba')" class="btn btn-primary b

şeklinde bir şey yazarsanız butona tıklandığında merhaba yazısını görürsünüz. alert k

Başka Bir Örnek:

<a href="#" id="todoClearButton" onclick="this.textContent = 'Buse'" class="btn btn-

Yazarsanız da butona tıklandığında butonun içindeki yazı Buse olarak değişir. Tek tır

Başka Bir Örnek:

<a href="#" id="todoClearButton" onclick="document.querySelectorAll('card-title')[1].

Yazarsak da card-title class'ına sahip ögenin içindeki yazıyı değiştirmiş oluruz. doc

Bu yöntemi fazla kullanmayacağız çünkü çok fazla JavaScript kodumuz olduğunda bu temi

index.html'e: <a href="#" id="todoClearButton" onclick="changeTitle()" class="btn bt
metotunu çalıştırmasını söylüyoruz. Bunu yaptıktan sonra app.js'e:

```
function changeTitle(){
   document.querySelectorAll('card-title')[1].textContent='Değiştirdik";
}
```

yazıyoruz. Bu şekilde kodumuz hem daha temiz hem de okunabilir oluyor. script src ola

Bundan daha da kullanışlı bir yöntem daha vardır o da addEventListener'ı kullanmaktır

const clearButton = document.querySelector("#todoClearButton"); ID'si todoClearButto

clearButton.addEventListener("click", function(){alert ("Merhaba!");}) addEventListe

Artık butona bastığımızda Merhaba yazısı çıkıyor.

Kısaca ekrandaki butonumuzu seçtik addEventListener kullanarak bu butona basıldığında

Bunun yerine daha önce tanımladığımız changeTitle fonksiyonunu addEventListener olara

```
const clearButton = document.querySelector("#todoClearButton");
clearButton.addEventListener("click", changeTitle);
```

Kısaca anlatmak gerekirse butona bir addEventListener ekledik ve click olma durumunda

Uyarı: changeTitle() şeklinde parantezli kullanırsanız addEventListener'ı beklemeden

Tüm eventleri konsolunuzda görmek için:

```
const clearButton = document.querySelector("#todoClearButton");
clearButton.addEventListener("click", changeTitle);

function changeTitle(e){
   console.log(e);
}
```

Yazabilirsiniz. Daha kolay yöntemleri de tabiki olabilir fakat bu kullanımı öğrenmeni

```
function changeTitle(e){
   console.log(e.target);
}
```

Yazarsanız da bu elemente tıklandığında hangi HTML etiketlerinin kullanıldığını görme

```
function changeTitle(e){
   console.log(e.target.textContent);
}
```

Yazarsanız dönen HTML elementinin içindeki yazıyı almış olursunuz.

```
function changeTitle(e){
   console.log(e.target.className);
}
```

Yazarak da istediğiniz herhangi bir elementin(biz burda butonu seçtik) class(sınıf) i

```
function changeTitle(e){
   console.log(e.type);
}
```

Yazarsanız hangi koşulda bu fonksiyonun çalışacağını konsolda görebilirsiniz. Bunu ya

MOUSE (Fare) Events

DOMContentLoaded
load

Yukarıdakilerin ikisi de sayfa yüklenmeyle ilgilidir.

click
dblclick
mouseover
mouseout
mouseenter
mouseleave

1) DOMContentLoaded

```
document.addEventListener("DOMContentLoaded", run);

function run{
   console.log("Sayfa Yüklendi...");
}
```

Yazarsanız sayfa yüklendiğinde direkt çalışır ve yazıyı size konsolda gösterir. Bunun

2) load

```
window.addEventListener("load", run);

function run{
   console.log("Sayfa Yüklendi...");
}
```

Yazarak da aynı işlemi yapabilirsiniz. Birinde window birinde document objesi kulland

3) click

const cardTitle = document.querySelectoAll("card-title")[1]; yazarak ilk önce card-ti
cardTitle.addEventListener("click", run);

```
function run(e){
   console.log(e.type);
}
```

Yazarsak seçilen butona tıklandığında konsolda ne kullandığımızı görebiliriz. Click k

4) dblclick

const cardTitle = document.querySelectorAll("card-title")[1]; yazarak ilk önce card-t
cardTitle.addEventListener("dblclick", run);

```
function run(e){
   console.log(e.type);
}
```

Yazarsak da sadece çift tıkladığımızda çalışır. İsterseni ayrı ayrı 10 kez basın yine

```
5) mouseover

const cardTitle = document.querySelector(".card-title")[1];
const cardBody = document.querySelectorAll(".card-body")[1];

cardBody.addEventListener("mouseover", run);

function run(e){
   console.log(e.type);
}

cardBody ve içindeki elementler üzerinde gezindiğimiz takdirde mousover çalışmaya dev

6) mouseout

const cardTitle = document.querySelector(".card-title")[1];
const cardBody = document.querySelectorAll(".card-body")[1];

cardBody.addEventListener("mouseout", run);
cardBody.addEventListener("mouseover", run);

function run(e){
   console.log(e.type);
}

Yazarsanız da cardBody dışına çıktığınızda çalışır. cardBody dışında başka bir elemen
cardBody'nin içine girdiğimizde mouseover dışına çıktığımızda mouseout çalışır. Başka

7) mouseenter

const cardTitle = document.querySelector(".card-title")[1];
const cardBody = document.querySelectorAll(".card-body")[1];

cardBody.addEventListener("mouseenter", run);

function run(e){
   console.log(e.type);
}

Bunu çalıştırdığınızda ise cardBody'nin üzerine geldiğinizde çalışır ama cardBody'nin
mouseover hem cardBody hem de cardBody içindeki HTML elementleri üzerinde gezindiğini

8) mouseleave

const cardTitle = document.querySelector(".card-title")[1];
const cardBody = document.querySelectorAll(".card-body")[1];

cardBody.addEventListener("mouseleave", run);

function run(e){
   console.log(e.type);
}
```

Bunu çalıştırdığınızda cardBody'nin dışına çıkarsanız çalışır. İçindeki HTML etiketle

Projenin birçok yerinde modifiye edilerek kullanılabilir bu yüzden bunları öğrenmemiz

KEYBOARD (Klavye) Events

keypress : Harf Ve Sayılarda Tetiklenir.
keydown : Hepsinde Çalışır.
keyup : Tuştan Elini Kaldırdığında Çalışır.

1) keypress

Bu sayfa içerisinde (document-todolist.html) klavyede herhangi bir tuşa basıldığında

```
document.addEventListener("keypress", run);  Butona basıldığında run metodu çalışacak

function run(e){
   console.log(e.type);
}
```

Bunu yazarak herhangi bir butona basıldığında sayfada hangi addEventListener tipinin
Tipimiz keypress olduğundan her butona bastığınızda konsolda keypress yazısını görece
Basmış olduğumuz tuşu yakalayıp ekranda göstermek istiyorsak:

```
document.addEventListener("keypress", run);  Butona basıldığında run metodu çalışacak

function run(e){
   console.log(e.key);
}
```

Yazarız ve ekrandayken A tuşuna basarsanız konsolda A'yı görebilirsiniz.

Eğer klavyeden bir tuşa basmak ve bu tuşun ASCII tablosundaki karşılığını görmek isti

```
document.addEventListener("keypress", run);  Butona basıldığında run metodu çalışacak

function run(e){
   console.log(e.keyCode);
}
```

Yazıp herhangi bir tuşa basarak ASCII tablosundaki karşılığını konsolda görebilirsini

keypress'in bir dezavantajı vardır o da klavyeden sadece harfleri ve sayıları algılay

2) keydown

```
document.addEventListener("keydown", run);  Butona basıldığında run metodu çalışacak.

function run(e){
   console.log(e.keyCode);
}
```

keydown keypress'in aynısıdır tek fark keydown'un klavyedeki her şeyi algılayabilmesi

3) keyup

document.addEventListener("keyup", run); Butona basıldığında run metodu çalışacak.

```
function run(e){
   console.log(e.keyCode);
}
```

keyup ise tuştan elimizi kaldırdığımızda çalışır. Örneğin ekrana geldiğinizde a tuşun

Örneğin kullanıcı klavyeden F5'e bastığında sayfa yenilemeyi engellemek istiyorsak:

document.addEventListener("keydown", run); Butona basıldığında run metodu çalışacak.

```
function run(e){
   console.log(e.keyCode);
   if(e.keyCode==116){   //116 yazmamızın sebebi F5'in ASCII kod tablosunda değerinin
       alert("Sayfa Yenileme Engellendi.");
   }

   e.preventDefault();  yazarak da engelleme işlemimizi tamamlamış oluruz.
}
```

Başka bir örnek yapalım input'a bir şeyler girelim ve Todo List başlığına girdiğimiz

const cardTitle = document.querySelectorAll(".card-title")[0]; yazarak 0. indexteki

cardTitle'ı seçmemiz yeterli değil input'un içine bir şey yazılma durumunu seçmek ist

const input = document.querySelector("#todoName"); yazarak input'u seçiyoruz.

input.addEventListener("keyup",run); yazıyoruz. Dikkat edin document'a değil input'a

```
function run(e){
   console.log(e.target.value);  yazarak input'un içindeki değeri alıyoruz.
}
```

Şuan bunu çalıştırırsanız input içine ne girerseniz girin konsolda görebilirsiniz yan

```
function run(e){
   cardTitle.textContent = e.target.value;
}
```

Yazarsak da amacımıza ulaşmış oluruz. Input'un içine ne yazarsak başlığımız o olur.

INPUT Events

focus
blur

```
copy
paste
cut
select
```

1) focus

focus imleci input içerisine bıraktığınızda ve inputa tıkladığınızda çalışan işleme d

Örneklerle kullanabilmek için sayfamızdaki (document-todolist.html) input'u seçelim:

const todo = document.querySelector(#todoName); yazarak seçelim başarılı bir şekilde

```
todo.addEventListener("focus", run);

function run(e){
   console.log(e.type);
}
```

yazarak focus her çalıştığında konsolda Event Listener tipini yazdırabiliriz.

2) blur

blur ise input'tan çıktığınızda çalışır yani focusladınız ve sonra çıktınız bu durumd

const todo = document.querySelector(#todoName); yazarak seçelim başarılı bir şekilde

```
todo.addEventListener("blur", run);

function run(e){
   console.log(e.type);
}
```

Bunlar gereksiz gibi gözükebilir ama projenin akışına göre kullanabileceğinizi ve işi

3) copy

copy adından da anlaşıldığı gibi input'un içindeki bir veriyi kopyalarsanız copy even

const todo = document.querySelector(#todoName); yazarak seçelim başarılı bir şekilde

```
todo.addEventListener("copy", run);

function run(e){
   console.log(e.type);
}
```

Kopyaladığınız, yapıştırdığınız her veri websiteleri tarafından alınabilir bu yüzden

4) paste

yine adından da anlayabileceğiniz gibi herhangi bir şeyi yapıştırırsanız konsolda bun

const todo = document.querySelector(#todoName); yazarak seçelim başarılı bir şekilde

```
todo.addEventListener("paste", run);

function run(e){
   console.log(e.type);
}
```

5) cut

yine adından da anlayabileceğiniz gibi herhangi bir şeyi keserseniz konsolda bu işlem

const todo = document.querySelector(#todoName); yazarak seçelim başarılı bir şekilde

```
todo.addEventListener("cut", run);

function run(e){
   console.log(e.type);
}
```

6) select

adından da anlayabileceğiniz gibi herhangi bir şeyi seçerseniz konsolda bu işlemi gör

const todo = document.querySelector(#todoName); yazarak seçelim başarılı bir şekilde

```
todo.addEventListener("select", run);

function run(e){
   console.log(e.type);
}
```

Session Storage Kullanımı

UYARI : Bu kısım biraz uzun olacak o yüzden mola vererek anlayarak ilerlemenizi tavsi

Tarayıcılarımızda 2 storage(depolama) tipi var biri local storage diğeri ise session

Sayfada inspect(incele) kısmına geldiğimizde application adlı bir bölüm var. Applicat
Bunların içine girdiğimizde key ve value mantığı var. İkisinin de kullanmış olduğu me
alanıdır. Kullanmış olduğunuz browser'ın depolama alanıdır.

Session Storage ve Local Storage arasındaki fark ise:

Session Storage'e herhangi bir değer ekledikten sonra tarayıcıyı kapatırsanız ya da b
bilgisayarı kapatsanız bile silinmez. İkisi arasındaki en büyük fark bu, başka bir fa

Storage verilerimizi depolamak için kullanmış olduğumuz bir hafızadır diyebiliriz.

console.log(window); yazarsanız Local Storage ve Session Storage'in window objesi içe

Direkt window.sessionStorage ya da window.localStorage yazarak erişebilirsiniz.

Session Storage - Değer Ekleme

sessionStorage.setItem("320", "Buse"); yaparak değer ekleyebiliriz 1 tane key bir ta
sessionStorage.setItem("193", "Talha");
sessionStorage.setItem("219", "Burak");

Yazarak incele kısmının Application > Session Storage'inden de görebileceğiniz gibi d

Session Storage - Değer Silme

sessionStorage.removeItem("320"); yazarak Buse isimli değeri Storage'den silebiliriz.

Session Storage - Değer Alma

sessionStorage içinden bir değeri alabiliriz:

```
let value = sessionStorage.getItem("320");
console.log(value);
```

Yazarak değeri alıp konsola yazdırabiliriz. Bunun yerine:

```
let value = sessionStorage.getItem("120");
console.log(value);
```

Yazsaydık key değeri 120 olan bir şey olmadığı için konsolda null yani boş değeri dön

Session Storage - Hepsini Silme

sessionStorage.clear(); Yazarsak da tüm değerleri sileriz.

Session Storage - Önemli Kısım

sessionStorage.setItem(249, 17);

Yazarsak ve bunun type'ına bakarsak:

```
let value = sessionStorage.getItems(249);
console.log(typeof value);
```

Bunun tipinin tırnaklar içerisinde yazmadığımız için int değil string olarak alındığı

Session Storage - Array Yazdırma

Örneğin isimlerden oluşan bir array'e sahip olalım:

```
let names = ["Buse","Burak","Berat","Talha"];
```

sessionStorage.setItem("names", names); ilk yazdığımız "names" değeri key'e eşit. İki

```
let value = sessionStorage.getItem("names");
console.log(value);
```

Kodlarını eklersek de key'i names olanların value değerlerini yani array içine yazdığ

```
console.log(typeof value);
```

tipinin array değil string olarak döndüğünü görebiliriz. Bunun tipinin array olarak d

```
sessionStorage.setItem("names", names);  yerine:

sessionStorage.setItem("names", JSON.stringify(names));
```

Yazmamız gerekiyor. Böyle yazarsak konsola array şeklinde yazdırmış oluruz.

Bunun tamamını bir array'miş gibi aldırmak istiyorsak:

```
sessionStorage.setItem("names", JSON.stringify(names));
let value = JSON.parse(sessionStorage.getItem("names"));
console.log(value);
```

Yazabiliriz. Daha iyi anlamak için tüm bunları deneyerek öğrenmeyi unutmayın. JSON.st
tamamen array gibi konsolda görmemizde yardımcı oldu. JSON.stringify ve JSON.parse me
JSON.parse kullanmasaydık üzerinde dönemeyecektik. Array gibi görünen string olmaması

```
let names = ["Buse","Burak","Berat","Talha"];
sessionStorage.setItem("names", JSON.stringify(names));

let value = JSON.parse(sessionStorage.getItem("names"));
value.forEach(function(name){
   console.log(name);
})
```

Yazarak da tüm array'deki isimleri konsola yazdırabiliriz. Daha iyi anlamak için kesi
Kısaca anlatmam gerekirse:
Storage'dan isimleri aldık sonrasında JSON.parse kullanarak bu isimleri array'e çevir
value'ya atadığımız değerin üzerinde döndük ve her birini name olarak yakalayıp konso

Eklemiş olduğumuz her şey bunları kullanmadan string tipinde alınır ve döner.

Son olarak setlerken JSON.stringify kullanıp array'miş gibi setlersiniz, ordaki array

NOT : Buraya kadar geldiysen helal olsun, disiplinli bir şekilde çalışmaya devam et.

Local Storage Kullanımı

Yukarıda bahsedilen fark hariç sessionStorage'dan hiçbir farkı yoktur. Kullanılan met
Kullanımı tamamen aynıdır.

Local Storage Değer Ekleme

```
localStorage.setItem("motion1", "push-up");
localStorage.setItem("motion2", "barfix");
localStorage.setItem("motion3", "burpee");
localStorage.setItem("motion4", "squat");
```

Local Storage Değer Almak

```
let value = localStorage.getItem("motion1");
console.log(value);
```

JSON.stringfiy kullanmama sebebimiz bu değerlerin bir array olmamamsı, array kullanma

Local Storage Değer Silmek

```
localStorage.removeItem("motion4");
```

Tümünü Temizle

```
localStorage.clear();
```

Bu Aşamadan Sonra Todo List Projesi Yaptık. Projeyi Görmek İçin todolistproje.html ve
Gerekirse videolardan izleyip ilerleyerek bir TODO List projesi yapın, alıştırma olma

EcmaScript (ES6) Nedir?

EcmaScript nedir sorusu hayli kafa karıştıran ve net bir cevabı olmayan bir sorudur.
EcmaScript bir standarttır. JavaScript'te bu standarı esas olan bir betik dilidir. Bu
bu dil üzerine konulmuş bir standarttır. Standart kelimesini şöyle açıklayabiliriz: Y
tüm tarayıcılarda çalışmasını sağlıyor. Kullanıcı deneyimi açısından çok önemli. Goog
kullanmak yeterli oluyor. EcmaScript'in birçok versiyonu vardır. JavaScript'le bir bü

* Arrow Functions

ES6 ile beraber önemli bir özelliktir. Önceden bir metot tanımlarken:

```
function yazdir(){
   consolelog("Merhaba");
}

yazdir()
```

şeklinde tanımlıyorduk fakat ES6'dan sonra şöyle bir yazım standartı geliştirildi:

ARROW FUNCTIONS

```
const yazdir = ()=>{
   console.log("Merhaba");
}

yazdir()
```

Function keyword'unu kullanmak yerine parantex açıp kapatıp eşittir ve büyüktür ifade
Bu bir metot aslında. fonksyionu alıp yazdir adlı değişkene atıyoruz.Çok da bir fark
Arrow functionlarda isimlendirme yoktur atama operatörü sayesinde değişkeni atar ve k
Bu fonksiyonu şu şekilde kısaltabiliriz:

Parametreli Arrow Function

```
const yazdir = (firstName)=>{
```

```
    console.log("Merhaba",firstName)
}

yazdir("Buse")
```

İkinci Parametreli Arrow Function Örneği:

```
const yazdir = (firstName, secondName)=>{
    console.log("Merhaba",firstName,secondName)
}

yazdir("Buse","Çetin")
```

Arrow functionların güzelliği yazdığınız kod satırı tek bir satırdan oluşuyorsa, yani oluşuyorsa kıvırcık parantezleri kaldırabilirsiniz:

```
const yazdir = (firstName, secondName)=>console.log("Merhaba",firstName,secondName)
yazdir("Buse","Çetin")
```

Yani tek satırda bir metot tanımı yapmanıza izin verir.
Tek satır değil de birden fazla satırdan oluşan kod bloğunuz var ise:

```
const yazdir = (firstName, secondName)=>
console.log("Merhaba",firstName,secondName)
yazdir("Buse","Çetin")
let a = 5;
console.log(a)
console.log("Buse")
yazdir("Buse","Çetin")
```

Gördüğünüz gibi istediğimiz çıktıyı vermedi çünkü JavaScript'te kodlar yukarıdan aşağ oluşan bir kod yazacaksanız, kıvırcık parantex kullanın.

Bunlar birçok JavaScript geliştiricisi tarafından kullanılan özelliklerdir. Bir kodu bunları sık sık kullanacaksınız.

```
const yazdir = firstName =>
    console.log("Merhaba", firstName)

yazdir("Buse")
```

Bir metot, bir arrow function'ın paramteresi tek bir metottan oluşuyorsa oval parante
Bunun anlamı şu burda yazdir adında bir function var ve bu function firstName adında
Arrow unction birçok yerde kullanılıyor, buna alışın çok karşılaşacağız.

```
const kupAl = x =>{
    return x*x*x
}

console.log = , kupAl(3);
```

Burda yaptığımız şey şu:

kupAl adındaki metodu çağırdık, x = 3 oldu ve 3*3*3 = 27 oldu sonrasında da bu 27'yi

* NOT *

İşlemimiz tek bir satırdan oluşuyorsa ve return kullanıyorsak hem oval paranteleri he

```
const kupAl = x => x*x*x

console.log = , kupAl(3);
```

Sadece buralarda değil Call Back Functionlarda da fazlaca kullanılır. Örneğin:

```
const kupAl = x => x*x*x

document.addEventListener("click", function(){

})
```

normalde böyle yazıyorduk fakat arrow function sayesinde şu şekilde yazabiliyoruz:

```
document.addEventListener("click", ()=>{

})
```

şeklinde kullanacağız. Gördüğünüz gibi arrow function kodu sadeleştirdiği gibi okumay

İlk başlarda kullanımı biraz zorlayabilir fakat kullandıkça ve pratik yaptıkca öğrene

Tanımlanmış olan arrow functionlar bir dğeişkene setlenir ve o değişken üzerinden çağ

```
console.log = , kupAl(3);
```

Destructing Kullanımı

ES6 ile gelen özelliklerimizden biridir. Kullanımı:

```
let langs = ["JS", "C", "C++", "Python"]
let lang 1, lan2, lang3, lang4

lang1 = langs[0];
lang2 = langs[1];
lang3 = langs[2];
lang4 = langs[3];

console.log(lang1,lang2,lang3,lang4);
```

Burda 1 tane array 4 tane de değişken tanımladık, arrayin her bir indexine değişkenle
Index numaralarını kullanarak değişkenleri doldurduk.
Normalde bunu böyşe yapıyorduk fakat Destructing sayesinde bu kullanım yerine daha ba

```
let langs = ["JS", "C", "C++", "Python"]
let lang 1, lan2, lang3, lang4
```

```
[lang1,lang2,lang3,lang4] = langs

console.log(lang1,lang2,lang3,lang4);
```

Burdaki langs'in 0. indexine bakıp bunu setliyor aynı şekilde diğer değişkenler için [] bu parantezleri kullanmamız gerekiyor. Kutu parantez dizinin göstergesidir. Gördüğünüz gibi kodumuz hem daha temiz hem daha okunabilir oldu. Kullanmasanız da olu

Bu konuyu daha iyi kavramak için w3schools'tan bakmanızı ve örnekleri denemenizi tavs

```
const hesapla = (a,b)=>{
   const toplam = a+b
   const cikar = a-b
   const carp = a*b
   const bol = a/b

   const dizi = [toplam,cikar,carp,bol]
   return dizi
}

let[a,b,c,d] = hesapla(18,2)

console.log(a,b,c,d)
```

Başka Bir Örnek

```
const person = {
   firstName : "Buse",
   lastName : "Cetin",
   salary : 5000,
   age : 18,
}

let isim, soyisim, maas, yas

isim = person.firstName
soyisim = person.lastName
maas = person.salary
yas = person.age

console.log(isim,soyisim,maas,yas)
```

şeklinde yapabiliriz fakat bunun daha kolay bir kullanımı vardır:

Not: Süslü parantez objeyi kutu parantez diziyi ifade eder.

```
const person = {
   firstName : "Buse",
   lastName : "Cetin",
   salary : 5000,
   age : 18,
}
```

```
{let firstName, lastName, salary, age} = person

console.log(firstName,lastName,salary,age)
```

Eğer değişkenleri firstName ya da lastName gibi isimlerle kullanmak istemiyorsanız:

```
{let firstName:isim, lastName:soyisim, salary:maas, age:yas} = person
```

şeklinde yazarak yeniden adlandırabilirsiniz. Fakat bunu yaptıktan sonra:

```
console.log(isim,soyisim,maas,yas)
```

Kısmını da yukarıdaki gibi değiştirmeniz gerek.

Spread Operatötü Kullanımı

Spread operatörünün ne işe yaradığını anlamak için önce spread kullanmazsak ne olur b

```
let numbers = [10,20,30,40]
function add(a,b,c,d){
   console.log(a+b+c+d)
}
```

Eski Yöntem:

```
add(numbers[0],numbers[1],numbers[2],numbers[3])
```

Yeni Yöntem:

```
add(...numbers)
```

Bunun Türkçesi şu : (numbers[0],numbers[1],numbers[2],numbers[3]) = (...numbers)

3 nokta spread operatörünü gösterir ki zaten spread'in anlamı dilim demektir. Diziyi

Örnek 2:

```
const diller1 : ["JavaScript", "Java"]
const diller2 : ["PHP", "Python"]
```

eğer diller2, diller1'i de kapsasın isteseydik:

```
const diller2 : ["PHP", "Python", diller1[0], diller1[1]]
console.log(diller2)
```

şeklinde yazacaktık. Bunun yerine daha kolay ve basit bir spread kullanımı da yapabil

```
const diller1 : ["JavaScript", "Java"]
const diller2 : ["PHP", "Python", ...diller1]
console.log(diller2)
```

Örnek 3: Farklı Kullanım

```
const numbers = [1,2,3,4,5,6,7,8,9]
let [a,b] = numbers
console.log(a,b)
```

yazarsak konsolumuzda sadece 0 ve 1. indexte olan 1 ve 2 rakamı gözükecektir, diğer r

```
const numbers = [1,2,3,4,5,6,7,8,9]
let [a,b, ...kalanSayilar] = numbers
console.log(a,b, kalanSayilar)
```

Yazarsak 1 ve 2 dışında kalan sayıları da konsolda görebiliriz. Yani 0. indexteki say

Örnek 4 : Arraydeki Değeri Başka Array'e Verme

```
const array1 = ["Buse", "Berat","Ali", "Burak"]
const array2 = []

array2[0] = array1[0]
array2[1] = array1[1]
array2[2] = array1[2]
array2[3] = array1[3]

console.log(array2)
```

Normalde yukarıdaki gibi yapıyorduk fakat spread kullanımı sayesinde artık çok daha k

```
const array1 = ["Buse", "Berat","Ali", "Burak"]


const array2 = [...array1]
console.log(array2)
```

yazarak yukarıda yazdığımız 4 satırlık kod bloğunu tek satıra indirebiliriz.

For in - For of Döngüleri

For in Kullanımı

for in bir dizi üzerinde dönerken o dizinin her bir elemanının indexini verir. for in

Normal Fonksiyon:

```
let names = ["Buse","Nur","Çetin","Emir","Hilal","Aybars"];

names.forEach(function(name){
   console.log(name)
})
```

Arrow Function:

```
let names = ["Buse","Nur","Çetin","Emir","Hilal","Aybars"];

names.forEach(name=> console.log(name))
```

For in kullanırken önce değişkeni tanımlıyoruz sonra let diyoruz ve sonra değişkene i

```
let names = ["Buse","Nur","Çetin","Emir","Hilal","Aybars"];

for(let name in names){
   console.log(name)
}
```

Yazarsanız names adlı dizinin her bir elemanının index numarasını görebilirsiniz faka
Değişken ismi verirken anlamlı, konuyla alakalı isim verin ki sizden sonra gelecek ol

```
for(let name in names){
   console.log(name, names[name])
}
```

Yazarak hem index numarasını hem o index numarasına karşılık gelen değeri konsolda gö

For of Döngüsü

Bir dizi üzerinde for of kullanıldığında direkt değerini verir. Değerinden indexine i

```
for(let isim of names){
   console.log(isim)
}
```

yazarsanız for in'de nasıl index numaralarını görüyorsak for of'da da değerleri yani

```
for(let isim of names){
   console.log(isim, names.indexOf(isim))
}
```

şeklinde yazmamız gerekiyor.

Map Kullanımı

Mapler arraylern alternatifidir. Key ve value mantığıyla çalışırlar. MAP = key(anahta

let array = [1,2,3] gibi dizileri oluşturuyorduk.

```
const map1 = new Map();   new OOP konusu fakat şimdilik bir map oluşturmak için kulla
```

Set

Map'in içerisine değer koymak için set metotu kullanılır. Map'in keyine de values'sun

```
const map1 = new Map();
map1.set(1,"Buse")
```

```
map1.set(true,5)
map1.set([1,2,3], {firstName: "Buse", lastName: "Çetin"})
map1.set(true, "5")

map1.set(16,"Bursa")
map1.set(34,"İstanbul")
map1.set(35,"İzmir")
map1.set(06,"Ankara")
```

Plakası 6 olanın değerini konsolda görmek istiyorsak get metodunu kullanırız.

GET

```
console.log(map1.get(6))
console.log(map1.get(06))
console.log(map1.get(35))
```

Yazarak plakaların içindeki değeri konsola yazdırabilirsiniz.

Size

Map'İn içinde ne kadar değer olduğu bilgisini size metotunu kullanarak alabiliriz.

```
let value;
map1.set(16,"Bursa")
map1.set(34,"İstanbul")
map1.set(35,"İzmir")
map1.set(06,"Ankara")

value= map1.size;      NOT: size bir property olduğu için size() olarak değil size ol
console.log(value);
```

Delete

Delete metotunu kullanarak mapin içinden değer silebiliriz.

```
let value;
map1.set(16,"Bursa")
map1.set(34,"İstanbul")
map1.set(35,"İzmir")
map1.set(06,"Ankara")

value = map1.delete(06)
console.log(value);
console.log(map1.size)
```

Yazarsak Ankara silinir. Konsolda silindiği için true görürsünüz. console.log(map1.si

HAS

Map'in içinde bir değer var mı diye sorgulama yapmamıza yarayan metottur. Bu map'in i

```
console.log(map1.has(16))
```

yazarsak ve konsolda true değerini görürsek bunun anlamı map'in içinde 16 plakalı değ

For Of Map Üzerinde Döndürme

Hangi keyin hangi value'ya denk geldiğini, tüm key ve value'ları görmek için kullanab

```
for (let [key, value] of map1){
   console.log(key,value)
}
```

Yukarıda aslında destructing kullandık.

Destructing öğrenirken:

```
let array = [34, 'İstanbul'];
let [a,b] = array;
console.log(a,b)
```

yazarak yukarıda yaptığımız işlemin aynısını yapıyorduk. Bu işlemle yukarıdaki işlem

Keys

Map'in içerisinde keys adında bir metot var ve bu metot key'leri bize geri döndürüyor

```
const keys = map1.keys();
console.log(keys)
```

Fakat bu keyler üzerinde forEach döngüsüyle dönemezsiniz. Ufak bir örnek yapalım:

```
const keys = map1.keys();
console.log(keys)
keys.forEach((key)=>{console.log(key)})  yazarsanız konsolda hata alırsınız. Çünkü bu

const keys = Array.from(map1.keys())
console.log(keys)

keys.forEach((key)=>{console.log(key)})
```

yazarak konsolda forEach kullanımınızı görebilirsiniz. Value ve keyi birlikte yazdırm

```
const keys = Array.from(map1.keys())

keys.forEach(key)=> {
   console.log(ey,map1.get(key))
}
```

Map'in içinden keyleri değil sadece değerleri almak istiyorsak ya da değerleri değil

Keyleri Almak İçin:

```
for(let key of map1.keys()){
   console.log(key)
```

```
}
```

Value'ları Almak İçin:

```
for(let value of map1.values()){
   console.log(value)
}
```

   Map'ten Array'e Çevirmek

Elimizde bir map varsa ve bunu array'e çevirmek istiyorsak kullanırız.

```
const arry = Array.from(map1);
console.log(array)
```

Bu array'in keylerini ve value'larını ufak bir kod bloğuyla istediğimiz zaman yazdıra

```
const array2 = [
   [16, "Bursa"],
   [34, "İstanbul"],
   [35, "İzmir"],
   [06, "Ankara"]
]

const array = Array.from(map1);

array.forEach((value)=>{
   console.log(value[0])
})
```

yazarsak keyleri görürüz index numarasına 1 verirsek de sadece value'ları konsolda gö

   Array'i Map'e Çevirmek

Bir map'i nasıl array'e çevirebiliyorsak array'i de map'e çevirebiliriz.

```
const array2 = [
   [16, "Bursa"],
   [34, "İstanbul"],
   [35, "İzmir"],
   [06, "Ankara"]
]

const myMap = new Map(array2);
console.log(myMap)
```

Önemli NOT:

```
map1.set(16,"Bursa")
map1.set(34,"İstanbul")
map1.set(35,"İzmir")
map1.set(06,"Ankara")
```

console.log(map1.get(6)) yaptığımızda Ankara geliyordu fakat aşağıdaki gibi bir kulla

```
map1.set([1,2,3], "Array")
console.log(map1.get([1,2,3]))
```

bunu yaparsanız hiçbir şey gelmez çünkü tanımladığımız tipler primitive tiplerken tan
tipleri çalıştığımız zamanları hatırlarsanız ikisi de farklı yere baktığı için birbir
Fakat bu şekilde kullanabilirsiniz:

```
let key = [1,2,3]

map1.set(16,"Bursa")
map1.set(34,"İstanbul")
map1.set(35,"İzmir")
map1.set(06,"Ankara")
map1.set(key, "Array")

console.log(map1.get(key))
```

Bunu bulabilme sebebi diziyi let ifadesiyle tanımlamamız. Primitive tiplerde tipine b

SET Kullanımı

Set'ler de Map'ler gibi dizilerin alternatifi olan bir dizi çeşitidir.
Set'lerin farkı : Set'lerin içerisine koyduğumuz değerler sadece 1 kez tutuluyor. Bir
Tek farkı 1 değeri yalnızca 1 kere tutabilmemiz. Set'lerde key-value mantığı yoktur.

Add Metodu

```
const set = new Set();
set.add(true)
set.add(3.14)
set.add("Buse")
set.add("Buse")
set.add("Buse")
set.add(7)
set.add({username:"buse" , passwrord : "0"});
set.add([1,2,3,4])
```

Dizimizin içine değerlerimizi koyduk. NOT: Set'in de bir dizi olduğunu unutmayın.

```
console.log(set.size)  yazarek set'in içinde kaç değer tuttuğunu görebilirsiniz. Görd
```

Delete Metodu

Set dizisinin içindeki herhangi bir değeri ya da değerleri silmek için kullanırız.

```
const set = new Set();
set.add(true)
set.add(3.14)
set.add("Buse")
set.add("Buse")
set.add("Buse")
set.add(7)
```

```
set.add({username:"buse" , passwrord : "0"});
set.add([1,2,3,4])

set.delete(true)
console.log(set.size)
```

Has

İstediğimiz değerin var olup olmadığını sorgulamak için kullanırız.

```
const set = new Set();
set.add(true)
set.add(3.14)
set.add("Buse")
set.add("Buse")
set.add("Buse")
set.add(7)
set.add({username:"buse" , passwrord : "0"});
set.add([1,2,3,4])

console.log(set.has("Buse"))
```

For Of Dönüsüyle Set Üzerinde Dönme

Set dizisi içerisinde dönmek için kullanılır.

```
const set = new Set();
set.add(true)
set.add(3.14)
set.add("Buse")
set.add("Buse")
set.add("Buse")
set.add(7)
set.add({username:"buse" , passwrord : "0"});
set.add([1,2,3,4])

for(let value of set){
   console.log(value)
}
```

Set'i Array'e Çevirmek

```
const set = new Set();
set.add(true)
set.add(3.14)
set.add("Buse")
set.add("Buse")
set.add("Buse")
set.add(7)
set.add({username:"buse" , passwrord : "0"});
set.add([1,2,3,4])

const values = Array.from(set);
```

```
console.log(values)
```

Farklı Yöntem:

```
const values = Array.from(set);
values.forEach((value)=>{
   console.log(value)
})
```

Array'den Set Oluşturmak

```
let array = [1, "Buse", true, "Berat", 15, [1,2,3]]

const newSet = new Set(array);
console.log(newSet)
```

Template Literals

String'leri birleştirmek için eskiden + operatörü kullanılırdı bu da hayli zordu. Tem stringleri birleştirme konusunda işimizi hayli kolaylaştırn bir özelliktir.

```
function write(firstName, lastName){
   console.log("İsim :" +firstName + " "+ "Soyisim :"+ lastName)
}
write("Buse Nur", "Çetin")
```

Burda 2 tane parametre verdik ve verdiğimiz parametreleri konsola yazdırdık. İki tane Template Literals'tan önce string birleştirme bu şekilde yapılıyordu.

Template Literals Kullanımı

ALT GR + İki Kez Noktalı Virgül Tuşu İle Tek Tırnak Yapabilirsiniz.

```
function write(firstName, lastName){
   console.log(`İsim: ${firstName} Soyisim: ${lastName}`)
}
write("Buse Nur", "Çetin")
```

Yazarak da aynı sonucu çok daha kısa bir kullanımla alabilirsiniz. Tek tırnak koyunca boşlukları algılıyor, dinamik olarak yazdırmak istediğimiz değerleri de dolar süslü p dinamik olarak da göndermek istediğimiz firstName ve lastName ifadelerini yazdık.

Statik : Değişmeyen
Dinamik : Değişen

NOT: Template Literals alt alta yazdırmayı da algılar.

```
function write(firstName, lastName){
   console.log(
       `
       İsim: ${firstName}
```

```
        Soyisim: ${lastName}
        `
        )
}
write("Buse Nur", "Çetin")
```

Yazarsanız başarılı bir şekilde konsolda alt alta yazdırmış olursunuz. Boşluğu algıla

Peki Bunlar Nerede Kullanılır?

İleriki derslerde asenkron kullanımı göreceğiz, bu derslerde REST API'a istekte bulun
Asenkron öğrenirken bu ve bu gibi durumlarda sıkça Template Literals kullanacağız. Li

Nesne Yönelimli Programlama (OOP) Nedir?

ES6 ile birlikte gelmiştir. JavaScript eskiden sadece frontend(önyüz) için kullanılan

C# gibi Java gibi gelişmiş değil temel düzeyde bir OOP yapısı vardır.

Peki nedir bu nesneye yönelik programlama?

Normalde yazılım amacı gerçek hayatı bilgisayarlara anlatmaktır diyebiliriz. Örneğin
10-15 tane ortak özelliği var. OOP'de ise örneğin bir class oluşturuluyor ve bu class
Her işlevin soyutlandığı bir yapı diyebiliriz. Gerçek hayatta gördüğünüz bir nesnenin

Daha Detaylı Açıklama İçin İnceleyiniz: https://www.argenova.com.tr/nesne-yonelimli-p

OOP Giriş - Nesne Oluşturma Ve Yapıcı Metot Kullanımı

OOP içinde class adında bir anahtar kelimemiz vardır. Bu anahtar kelimeyi kullanarak
NOT: İsimlerdirme yaparken Türkçe karakter kullanmamaya dikkat edin. Constructor = Ya

```
class Insan {
    constructor(){
        console.log("Buse - Yapıcı Metot Çalıştı")
    }
}

const insan1 = new Insan();   // Bu kısımda insan sınıfından bir obje oluşturduk. ins
```

Kısaca özet geçeyim. Önce class kullanarak bir sınıf oluşturduk sonra bu sınıfın için

```
class Insan {
    constructor(){
        console.log("Buse - Yapıcı Metot Çalıştı")
    }
}

const insan1 = new Insan();
const insan2 = newInsan();
```

2 kere nesne oluşturduk ve kodumuzu 2 kere yazılmış şekilde konsolda gördük.
Bir tane obje oluşturduğunuzda ram bellekte bir tane insan1 bir tane insan2 oluşuyor,

Peki Yapıcı Metot (Constructor) Nedir?

Bir sınıfın içinde özellikler, yapıcı metot, tanımladığımız fonksiyonlar olur. Yapıcı

```
class Insan {
   constructor(isim,soyisim,yas,maas){
       console.log("Buse - Yapıcı Metot Çalıştı")
   }
}

const insan1 = new Insan("Buse Nur", "Çetin", 18, 100);  // Yapıcı metota parametrele
```

Şimdi gelin yapıcı metot içerisine özellikler tanımlayalım.

```
class Insan {
   constructor(isim,soyisim,yas,maas){
       this.isim = isim;
       this.soyisim = soyisim;
       this.yas = yas;
       this.maas = maas
   }
}

const insan1 = new Insan("Buse Nur", "Çetin", 18, 100);
```

Bu şekilde yapıcı metot içine özellikler tanımlayabilir ve parametre kullanarak bu öz

Şimdi de nasıl constructor içine fonksiyon yazabileceğimize bakalım:

```
class Insan {
   constructor(isim,soyisim,yas,maas){
       this.isim = isim;
       this.soyisim = soyisim;
       this.yas = yas;
       this.maas = maas
   }

   / Fonksiyonlar class içine değil bu kısıma yazılır.

   bilgileriGoster(){
       console.log(
           `
           İsim: ${this.isim}
           Soyisim: ${this.soyisim}
           Yaş: ${this.yas}
           Maaş: ${this.maas}
           `
           )
   }

   NOT: Yukarıda Template Literals Kullandık.
}
```

```
const insan1 = new Insan("Buse Nur", "Çetin", 18, 100);
insan1.bilgileriGoster();  // Metotu çağırdık.
```

Başka bir insanın bilgilerini de aynı şekilde konsola yazdırabiliriz.

```
class Insan {
   constructor(isim,soyisim,yas,maas){
       this.isim = isim;
       this.soyisim = soyisim;
       this.yas = yas;
       this.maas = maas
   }

   bilgileriGoster(){
       console.log(
           `
           İsim: ${this.isim}
           Soyisim: ${this.soyisim}
           Yaş: ${this.yas}
           Maaş: ${this.maas}
           `
           )
   }
}
const insan1 = new Insan("Buse Nur", "Çetin", 18, 100);
const insan2 = new Insan("Berat", "Çetin", 23, 1000000)
insan1.bilgileriGoster();
insan2.bilgileriGoster();
```

Aynı kodları defalarca kez 2. kişi için de yazmak yerine 1 kere yazıp birçok kez aynı
Sadece farklı 2 obje kullanarak yapıcı metot sayesinde ikisindeki bilgilere de erişeb

NOT: Bir sınıfın içindeki değişkenlere, özelliklere, tanımlanan metotlara ve fonksiyo

```
console.log(insan1.isim)
console.log(insan2.isim)
```

yazarak insan1 ve insan2'nin yalnızca isimlerine de erişebilirsiniz. İsim özelliğine

Static Nedir

Statik isminin zaten duran, değişmeyen gibi bir anlamı var. Bazı değişkenlerimizi vey
Static, Yapıcı Metot ve OOP gibi konuları anlamak biraz zor olduğundan örnekler yapar

Örnek:

```
class Matematik{
   topla(a,b){
       console.log(a+b)
   }
   cikar(a,b){
       console.log(a-b)
   }
```

```
    carp(a,b){
        console.log(a*b)
    }
    bol(a,b){
        console.log(a/b)
    }
}

const islem = new Matematik();
islem.topla(10,4);
```

Burada sınıftan bir nesne türettik, referansımız üzerinden topla adındaki metota ulaş
Peki static bunun neresinde?

```
class Matematik{
    static topla(a,b){
        console.log(a+b)
    }
    cikar(a,b){
        console.log(a-b)
    }
    carp(a,b){
        console.log(a*b)
    }
    bol(a,b){
        console.log(a/b)
    }
}

const islem = new Matematik();
islem.topla(10,4);
```

Yazarsanız konsolda hata alacaksınız çünkü bir sınıfın içindeki metot ya da fonksiyon
Peki nasıl erişeceğiz?

Sınıf ismi üzerinden erişeceğiz :

```
class Matematik{
    static topla(a,b){
        console.log(a+b)
    }
    cikar(a,b){
        console.log(a-b)
    }
    carp(a,b){
        console.log(a*b)
    }
    bol(a,b){
        console.log(a/b)
    }
}

Matematik.topla(10,9)
```

NOT: Bir şey statik olarak tanımlanmamışsa nesne üzerinden erişilir, statik olarak ta
Static = Class üzerinden erişim
Static değil = Nesne üzerinden erişim

Bir function veya özellik statikse bu function veya özellik nesneye değil class'a özg

Peki biz bu statiği nerede kullanacağız?

Nesne türetmek maliyetli bir iş çünkü her nesne türettiğinizde bu RAM bellekte yer ka
bu class'ın içine metotlar yazarsınız, bu metotları statik olarak tanımlayıp proje iç
nesne türetip RAM bellekte fazlaca yer kaplayan bir uygulamaya sahip olacaktık.

Örnek :

```
class Insan{
   static languagesCount=10;

   constructor(firstName, lastName, salary){
       this.firstName = firstName;
       this.lastName = lastName;
       this.salary = salary;
   }

   writeInfo(){
       console.log(this.firstname,this.lastName,this.salary,this.languagesCount)
   }
}

const insan1 = new Insan("Buse Nur","Çetin", "1");
insan1.writeInfo();
```

Yazarsak static olarak tanımladığımız languagesCount değeri bize konsolda undefined o
değerlerine erişebilirken statik olarak tanımlanan languagesCount değeri undefined dö
Bunu görmek için aşağıdaki kodu yazarız:

```
console.log(Insan.languagesCount);
```

Bir şey static olarak tanımlanmazsa nesne türeterek erişiriz, statik olarak tanımlana
sınıftan erişemezsiniz, static olarak tanımlanırsa da nesne üzerinden erişemezsiniz.

Asenkron Yapısına Giriş

1- JavaScript senkron çalışan bir programlama dilidir. Yukarıdan aşağıya derlenir ve

2- JavaScript bazı durumlarda asenkron çalışır. JavaScript'in asenkron çalıştığı zama

Web API tarafından yönetilen her şey asenkron çalışır.

Kısaca JavaScript senkron çalışan bir programlama dili her şey sırayla çalışıyor ama

Timing Kullanarak Asenkron Çalıştırma

Örnek 1:

```
console.log("Buse Nur");

setTimeout(()=>{
   console.log("Süre doldu ve çalıştı.")
}, 1000)

console.log("Çetin");
```

Örnek 2:

```
console.log("Buse Nur");

setTimeout(()=>{
   console.log("1000ms süre doldu ve çalıştı.")
}, 1000)

setTimeout(()=>{
   console.log(" 500ms süre doldu ve çalıştı.")
}, 500)

console.log("Çetin");
```

Örnek 3:

```
console.log("Buse Nur");

setTimeout(()=>{
   console.log("1000ms süre doldu ve çalıştı.")
}, 1000)

setTimeout(()=>{
   console.log(" 500ms süre doldu ve çalıştı.")
}, 500)

setTimeout(()=>{
   console.log(" 500ms süre doldu ve çalıştı.")
}, 750)

console.log("Çetin");
```

Konuyu daha iyi anlamak için bu örnekleri yazıp konsol çıktısını inceleyiniz.

Asenkron programlama ne kadar güzel olsa da önce Buse'yi sonra Çetin'i sonra 1000ms o

CALLBACK - PROMISE - ASYNC & AWAIT

Asenkron programlamayı yönetmek için kullanılır. Asenkron yapıları senkrona çevirip y

Asenkron Problemi

İlk başta user ID'yi bulacağız sonra user ID'si 5 olanı yakalamaya çalışacağız.

```
const users = [
   {
       userId: 5,
       post : "Buse Post - 1"
   },
   {
       userId: 5,
       post : "Buse Post - 1"
   },
   {
       userId: 5,
       post : "Buse Post - 1"
   },
   {
       userId: 6,
       post : "Berat Post - 1"
   },
   {
       userId: 7,
       post : "Burak Post - 1"
   },
]

function getUserId(){
   setTimeout(()=> {
       Servise gittik ve cevabı aldık.
       return 5;
   }, 1000)
}

function getPostByUserId(userId){
   Gerçek bir Rest API'a istek attığımızı varsayalım.
   setTimeout(()=> {
       users.forEach((user)=> {
           if(user.userId===userId){
               console.log(user.post);
           }
       })
   }, 500)
}

let userId = getUserId();
getPostByUserId(userId);
```

İlk başta user Id metotuna gitti 1 saniye bekledikten sonra 5 değerini döndü sonrasın
Bunun çalışmama sebebi setTimeout metotunun asenkron çalışmasıdır. Her iki metotta as

Bu problemi çözebilmek için asenkronu senkrona çevireceğiz. Bunu yapmak için de CALLB

Nedir Bu CALLBACK

En eski yöntem callback'lerdir. Çok eski bir yöntem. Kullanılmıyor, kullanmanızı da t

Süre bakımından dezavantajlı olsa bile çalışma mantığı açısından karda olacağız.

```
function getName(){
   setTimeout(() => {
       servisten ismi getirdi varsayalım.
       console.log("Buse Nur");
   }, 1000)
}

function getSurname(){
   setTimeout(() => {
       console.log("Çetin");
   }, 500)
}

getName();
getSurname();
```

İlk önce getName'i çağırıp isim almak istememize rağmen asenkron çalıştığı için önce

* Callback Kullanarak Asenkronu Senkrona Çevirmek *

CALLBACK: Bir fonksiyonu bir fonksiyona parametre geçerek asenkron yapıyı senkrona çe

Örnek 1:

Öncelikle bir paramtere alacağız sonra soyisim fonksiyonunu isim fonksiyonu içine koy
Parametre olarak callback ismini verdim fakat siz Ayşe Ali Veli gibi isimler kullanab

```
function getName(callback){
   setTimeout(() => {
       servisten ismi getirdi varsayalım.
       console.log("Buse Nur");
   }, 1000)
}

function getSurname(){
   setTimeout(() => {
       console.log("Çetin");
   }, 500)
}

getName(getSurname);
```

Böyle yaptığımızda parametre olarak verdiğimiz callback = getSurname oluyor.

* Arrow Function ve Callback Kullanarak İsimsiz FOnksiyon Verme *

```
function getName(callback){
   setTimeout(() => {
       let name = "Buse";
       callback(name);
   }, 1000)
```

```
}

function getSurname(name,callback){
   setTimeout(() => {
       let surname = "Çetin";
       callback(surname)
   }, 500)
}

getName((name)=> {
   getSurname(name,(surname)=>{
       console.log(name, surname)
   }
})
```

Callback Hell yazıp internetten aratarak niye callback'leri kullanmayacağımızı daha i
gördüğünüz gibi Callback'ler okunabilirliği çok fazla azaltıyor.

Yukarıdaki örnekte kişinin ismini bulduk sonra ismini kullanarak soyismini bulduk. Şi

```
function getName(callback){
   setTimeout(() => {
       let name = "Buse";
       callback(name);
   }, 1000)
}

function getSurname(name,callback){
   setTimeout(() => {
       let surname = "Çetin";
       callback(surname)
   }, 500)
}

function getAge(name, surname, callback){
   setTimeout(()=> {
       let age = 18;
       callback(age)
   }, 300)
}

getName((name)=> {
   getSurname(name,(surname)=>{
       getAge(name, surname, (age)=> {
           console.log(name, surname, age);
       })
   }
})

  * Daha Önce 'Undefined' Aldığımız Hatanın CALLBACK Çözümü *

function getUserId(callback){
   setTimeout(()=> {
```

```
        Servise gittik ve cevabı aldık.
        return 5;
    }, 1000)
}

function getPostByUserId(userId){
    Gerçek bir Rest API'a istek attığımızı varsayalım.
    setTimeout(()=> {
        users.forEach((user)=> {
            if(user.userId===userId){
                console.log(user.post);
            }
        })
    }, 500)
}

let userId = getUserId();
getPostByUserId(userId);
```

Undefined Aldığımız Kod Buydu

Lütfen ilk önce kendiniz yapmaya çalışın.

Çözüm

```
const users = [
{
    userId: 5,
    post : "Buse Post - 1"
},
{
    userId: 5,
    post : "Buse Post - 1"
},
{
    userId: 5,
    post : "Buse Post - 1"
},
{
    userId: 6,
    post : "Berat Post - 1"
},
{
    userId: 7,
    post : "Burak Post - 1"
},


]


function getUserId(callback){
setTimeout(()=> {
```

```
      let userId=5;
      callback(userId);
  }, 1000)
   }

    function getPostByUserId(userId){
       setTimeout(()=> {
           users.forEach((user)=> {
               if(user.userId===userId){
                   console.log(user.post);
                   return userId;
               }
           })
       }, 500)
    }

    getUserId(getPostByUserId);
```

AJAX Nedir?

AJAX'ın açılımı "Asynchronous JavaScript And XML"'dir. AJAX bir programlama dili değ

Daha iyi anlamak için W3 Schools'tan "What is AJAX" yazısını okumanızı tavsiye ederi kullanılıyor gibi düşünebilirsiniz.

AJAX Kullanımı

AJAX'ın içinde bir objemiz var ve biz bu objenin class'ından nesne türetiyoruz. Kons

```
cont xhr = new XMLHttpsRequest();
console.log(xhr);
```

Bunların içerisinde 3 önemli özellik var.

1- Status
Server'a istek attıktan sonra başarılı olursa 200, bulunamadıysa 403, sayfa yoksa

2- Ready State
0- Daha request(istek) atılmadı.
1- Sunucu bağlantısı kuruldu.
2- Request ulaştı.
3- Request işleniyor.
4- Request bitti.

Ready State 4 ise cevap hazırdır. Cevap hazırsa responseText özelliğini kullanırız.

3- onreadystatechange

Ready state 0 sonrasında 1 - 2 - 3 - 4 olarak sürekli değişiyor. Bu değişiklik ol onreadystatechange kullanırız. Daha iyi anlamak için bıraktığım linki inceleyiniz

Karşıdan veri almak için GET http tipini kullandıktan sonra veri almak istediğimi server'a gönderiyoruz, serverdan gelen cevabı da alıyoruz. Eski bir yöntemdir çok

AJAX Kullanımı

AJAX, daha önce de belirttiğim gibi uygulamamızdan herhangi bir server'a gidip or

AJAX Örnekli Anlatım

Bu kısımda AJAX nasıl kullanılır? sorusuna kodlayarak cevap verilecek fakat bu kı
Link: https://www.hosting.com.tr/bilgi-bankasi/rest-api/

Kısaca REST API'lar bir web servisidir, biz http protokolleri üzerinden web servi

Örnek için kullanılan websitesi: https://jsonplaceholder.typicode.com/

Örnek:

İlk başta yorumları getirelim. Bir URL bir ID alsın. Http isteği başarılı olup ge
URL'i hazırlamak için bir metot yazalım. Bu metot sayesinde ID'nin boş olup olmam
JSON'a çevrilmiş olarak kullanmak istediğimiz için de JSON.parse metotunu kullanı

```
function prepareURL(url, id){
   if(id===null){
       return url;
   }
   return `${url}?postId=${id}`
}

function getComments(url, id){
   let newURL = prepareURL(url, id);

   const chr = new XMLHttpsRequest();
   xhr.addEventListener("readystatechange", ()=>{
       if(xhr.readyState===4 && xhr.statıs===200){
           console.log(JSON.parse(xhr.responseText));
       }
   })

   xhr.open("GET", newURL)
   xhr.send();
}

getComments("https://jsonplaceholder.typicode.com/comments")
```

PROMISE

Buraya kadar mola vermediyseniz 1 saat mola vermenizi tavsiye ederim çünkü bu kon

PROMISE'ler CALLBACK'lerin alternatifi olarak kullanılan yapılardır. CALLBACK'ler
yönetebileceğiz. Promise'ler 3 aşamadan oluşur. 1- Pending(Bekleme) 2-Fullfiiled(

Bir isteğ atıyoruz, isteği attıktan sonra zaten bekleme(pending) modundayız. Eğer
olur, yani işlem başarılıdır. İşlem başarısız olursa reject olur.

Başarılı ve başarısız olma durumunu yönetebilmemiz için .then ve .catch adında 2

Sonraki notlarda anlamadığınız bir yer olursa bıraktığım linkten daha detaylı inc

İlk başta arrow function kullanarak bir promise oluşturalım. Bu promise resolve v

```
const promise1 = new Promise((resolve, reject)=>{

})
```

Daha sonra bu promise için belli kontroller sağlayarak konsola yazdıralım:

```
let check = true;

const promise1 = new Promise((resolve, reject)=>{
   if(check){
       resolve("Promise Başarılı");
   }else{
       reject("Promise Başarısız");
   }
})

 console.log(promise1);
```

Bu isimlendirmeleri bilmeniz iyi olur çünkü çoğu yazılımcı bu şekilde veya benze

Burdaki mantık şöyle: Başarılı olduğunda resolve adındaki function'ı tetiklerken
Promise'in içindeki functionları görebilirsiniz. Bunların kullanımını birazdan y
Promise başarılı yazısı resolve'un içinden promiseResult'a geliyor. let check =

Bir promise'imiz varsa 2 tane durumu var. Başarılı olursa resolve, başarısız olu

PROMISE Yakalama

Şuana kadar Promise nedir? Nasıl kullanılır? Bunları öğrendik ama bunları .then

```
let check = true;

function createPromise(){
   return new Promise((resolve, reject)=>{
       if(check){
           resolve("Promise Başarılı Oldu");
       }else{
           reject("Promise Başarısız Oldu");
       }
   })
 }

createPromise()
```

Değeri yakalayabilmek için noktalı virgül koymadan .then yazıp arrow function kul
Başarısız olma durumu içinse .catch kullanacağız.

```
let check = true;
```

```
function createPromise(){
   return new Promise((resolve, reject)=>{
       if(check){
           resolve("Promise Başarılı Oldu");
       }else{
           reject("Promise Başarısız Oldu");
       }
   })
 }

createPromise()
.then((response)=>{
   console.log(response);
})
.catch((error)=>{
   console.log(error)
})
```

Bu çok önemli çünkü sürekli kullanacağız bu yüzden deneyerek öğrenmenizi tavsiye

.catch ve .then dışında .finally adında bir function'ımız var, bu da bir arrow fu
.finally başarılı da başarısız da olsa çalışan bir fonksiyon. Her halükarda çalış

```
let check = true;

function createPromise(){
   return new Promise((resolve, reject)=>{
       if(check){
           resolve("Promise Başarılı Oldu");
       }else{
           reject("Promise Başarısız Oldu");
       }
   })
 }

createPromise()
.then((response)=>{
   console.log(response);
})
.catch((error)=>{
   console.log(error)
})
.finally(()=> console.log("Her Koşulda Çalışır."))
```

let check = true; kısmını let check = false; olarak yazarsanız her iki durumda da

Promise'leri de Callback'ler gibi asenkron yapıları senkrona çevirmek için kullan
ES6 ile beraber hayatımıza girdi, hala büyük projelerde kullanılıyor. Callback'le

Asenkron Yapı Oluşturup PROMISE Kullanarak Senkrona Çevirmek ∗ ∗ PROMISE + XML

Promise nedir? Ne değildir? Nasıl kullanılır? Az çok öğrendik. Şimdi de PROMISE k

senkrona çevirebileceğimize bakalım, önceki derste CALLBACK'ler ile yaptığımız şe

Bunu yapmadan önce kendi localimizdeki dosyayı çekmeye çalışalım. Try-catch kulla
try-catch JavaScript yazıp mutlaka göz atın. Hatalara try-catch kullanarak bakaca

```
function readStudents(url){
     return new Promise((resolve, reject)=>{
     const xhr = new XMLHttpRequest();
     try {
     xhr.addEventListener("readystatechange", ()=>{
        if(xhr.readyState===4 && xhr.status===200){
        resolve(JSON.parse(xhr.responseText));
     }
     })
    } catch (error) {
          console.log("JSON'da Problem Var.")
          reject(error);
     }
     xhr.open("GET", url);
     xhr.send();
     })}
     readStudents("students.json")
     .then((data)=> console.log(data))
     .catch((err)=>console.log(err))
```

Birden Çok Promise Tanımlamak

Promise.all() = Tanımlanan promise'leri dizi olarak alarak hepsini seçmenizi sağl
1 tanesi bile hata alırsa reject eder.

Birden Çok Promise Tanımlamak - Örnek

```
const pl = Promise.resolve("Birinci promise başarılı.")
const p2 = Promise.resolve("İkinci promise başarılı.")
const p3= new Promise((resolve, reject) =>{
resolve("Üçüncü promise başarılı.")
})
const p4 = Promise.reject("Hata var.");

Promise.all([pl,p2,p3,p4])
 .then((res)=>{
console.log(res)
 })
.catch((err) => console.log(err))
```

FETCH API

Fetch API, server'a istek atmak için kullandığımız window objesi içinde yer ala
Fetch API, promise ile beraber çalışıyor. Fetch API içine isteğimizi atıyoruz,
Fetch API geriye PROMISE döner.

```
function getStudents(url){
```

```
 fetch(url)
 .then((response)=>{
     return response.json();
 })
 .then((data)=> console.log(data))
 .catch((err) => console.log(err))
}

getStudents("students.json")
```

JSON almak istediğimiz veriyi almak için kullanılır. İlk başta promise'leri ret
yakaladık. Dönen response tipinden dolayı 2 kere .then ile yakaladık.

Arrow function'dan hatarlarsanız kod tek satırdan oluşuyorsa return yazmanıza v
Bunun için bir örnek yapalım:

```
function getData(url){
 fetch(url)
 .then((response)=> response.json());
 .then((data)=> console.log(data))
 .catch((err) => console.log(err))
}

getData("https://jsonplaceholder.typicode.com/comments");
```

Fetch API'ı bu şekilde kullanmak yerine başka bir şekilde de kullanabiliriz:

```
function getData(url){
 return fetch(url)
}

getData("https://jsonplaceholder.typicode.com/users");
.then((response)=> response.json())
.then((data)=>console.log(data))
.catch((err)=>console.log(err))
```

Başka Bir Yöntem:

```
function getData(url){
 const promise = fetch("https://jsonplaceholder.typicode.com/albums")
 console.log(promise)
}

getData("https://jsonplaceholder.typicode.com/users");
```

  POST Kullanma

Şuana kadar hep GET kullanarak veri aldık şimdi de POST kullanarak veri yükleye

```
function saveStudents(){
 fetch(https://jsonplaceholder.typicode.com/users),{
     method: "POST",
     headers : {
```

```
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            "id" : 1,
            "firstname": "Buse",
            "lastname": "Çetin"
        })
 }
}
```

Fetch adresine POST metot tipiyle application/json tipinde değer vereceğimizi v
yükleyeceğimizi belirttik. Yükledikten sonra başarılı olup olmadığını .then ve

ASYNC AWAIT

ASYNC AWAIT, promise'lerden ve callback'lerden çok daha gelişmiş bir yöntemdir.

ASYNC AWAIT Olmadan

```
document.querySelector("#button").addEventListener("click",()=>{
 fetch("https://jsonplaceholder.typicode.com/posts/1")
 .then((response)=> response.json())
 .then((post)=>{
     console.log(post)
     fetch(`https://jsonplaceholder.typicode.com/comments?postId=${post.id}`)
     .then((response)=> response.json())
     .then((comments)=> console.log(comments))
 })
})
```

ASYNC AWAIT Kullanarak

```
document.querySelector("#button").addEventListener("click",async ()=>{
 const post = await (await fetch ("https://jsonplaceholder.typicode.com/posts/1
 const comments = await(await fetc(`https://jsonplaceholder.typicode.com/commen
 console.log(post,comments)
})
```

İlk önce fetch'le adresimizi yazdık. Fetch bize promise tipinde dönüş yapar son
ID'si 1 olan postun yorumlarını getireceğiz. Bir tane daha fetch açtık ve dinam

Kısaca bir daha üzerinden geçelim. Ekrandan ID'si button olan butona basınca ar
aldık ve gelen post promise ile döndüğü için .then ile yakaladık, sonrasında re
Sonrasında dönen promise'i bir sonraki .then ile yakaladık. Sonrasında dinamik

Await yalnızca asenkron fonksiyonlarda kullanılabilir bu yüzden await kullanabi
Yani eğer bir yerde await kullanıyorsanız, kullanmış olduğunuz fonksiyonun başı
kod çalıştıktan sonra 2. satırdaki kod çalışsın yani asenkron problemi oluşması

Fetch API ile bir yere istek attığınızda bu asenkron çalışıyor. Asenkron proble

Gördüğünüz gibi ASYNC Await ile kod hem daha kısa hem daha okunabilir oluyor am
bilmemiz gerek.

Asenkron Bölümü Özet

Senkron: Sırayla çalışan işlem parçacıkları.
Asenkron: Eş zamanlı birden fazla iş yapmak için kullanılır.

JavaScript senkron çalışan bir programlama dilidir.

Asenkron Çalışmasına Sebep Olanlar

1- Timing(Zaman)
2- Event(Olay)
3- HTTP İstekleri
- XmlHttpRequest
- Fetch API
- Axios

Asenkronu Yönetmek İçin

1- Callback - ES6 - Öncesi
2- Promise - ES6 - 2015
3- ASYNC AWAIT - ES7

Bu kısıma kadar gelebilecek özveriyi gösterdiğin için seni tebrik ediyorum. U
proje ve pratik yaparak kendini geliştirmeye, anlamadıklarını öğrenmeye devam