JAVASCRİPT

1990 yıllarının başında HTML diline destek olması için Netscape çalışanı Brendan Eich tarafından üretilen bir script yazılım dili olan JavaScript, 1997 yılından itibaren bir ECMA European Computer Manufacturers Association, Avrupa Bilgisayar Üreticileri Birliği standardı olan JavaScript, günümüzde Mozilla Vakfı öncülüğünde özgür yazılım topluluğu tarafından geliştirilmekte ve bakımı yapılmaktadır.

Javascript’in resmi adı ECMAScript’dir.

Html, Css, Js betik dildir. Betik dili, bir programlama dilinin basitleştirilmiş bir versiyonudur

Fonksiyonlar, bir işlevi yerine getirmek için tasarlanmış kod bloklarıdır.

JavaScript kendi DOM yapısı sayesinde HTML ve CSS yapılarını düzenleme imkanına sahiptir.

JavaScript’nin amacı, web sitelerini etkileşimli hale getirmektir. Aynı zamanda HTML, CSS ve PHP’nin aksine sunucu yerine kullanıcının internet tarayıcısında çalışır.

İstemci taraflı çalışma:

Kullanıcının web tarayıcısında gerçekleşir.

HTML, CSS ve JavaScript gibi web teknolojilerini kullanır.

Kullanıcı ile web sayfası arasında etkileşimi sağlar.

Örnekler:

Bir butona tıklama

Bir form doldurma

Bir animasyon oynatma

Bir video izleme

Sunucu taraflı çalışma:

Web sunucusunda gerçekleşir.

PHP, Python, Java gibi programlama dillerini kullanır.

Veritabanlarına erişim ve veri işleme gibi işlemleri gerçekleştirir.

Örnekler:

Bir kullanıcının giriş bilgilerini doğrulama

Bir ürün satın alma işlemini gerçekleştirme

Bir e-posta gönderme

Bir web sayfasının içeriğini oluşturma

**JavaScript'in çalışma prensibi:**

JavaScript kodu HTML dosyasına gömülür.

Web tarayıcısı JavaScript kodunu yorumlar.

Kod, tarayıcı tarafından bir dizi adıma bölünür:

Tokenization: Kod, kelimeler ve semboller gibi daha küçük parçalara ayrılır.

Parsing: Kelimeler ve semboller, bir anlam ağacı oluşturmak için bir araya getirilir.

Code Generation: Anlam ağacı, tarayıcının anlayabileceği makine koduna dönüştürülür.

Makine kodu, tarayıcı tarafından yürütülür.

Sonuç, web sayfasında görüntülenir.

**Java'nın çalışma prensibi:**

Java kodu bir metin editöründe yazılır.

Kod, javac komutuyla derlenir.

Derleme işlemi, Java kodunu bytecode adı verilen bir makine koduna dönüştürür.

Bytecode, Java Sanal Makinesi (JVM) tarafından yürütülür.

JVM, her işletim sistemi için özel olarak yazılmış bir programdır.

**Java Sanal Makinesi (JVM) Nasıl Çalışır?**

Java Sanal Makinesi (JVM), Java bytecode'ı çalıştıran sanal bir makinedir. Java kodunun platformdan bağımsız çalışmasını sağlar.

JVM'nin çalışma prensibi:

Java kodu derlenir ve bytecode'a dönüştürülür.

Bytecode, JVM tarafından yürütülür.

JVM, bytecode'u işletim sistemine özel makine koduna dönüştürür.

Dönüştürülen kod, işlemci tarafından yürütülür.

Escape karakter: metin içinde özel anlamlı karakterlerin kullanılmasını sağlar. Escape karakteri genellikle ters bölü çizgi (\) ile ifade edilir.

Var ismim = “Yusuf \”sefa\” sezer”; alert(ismim) output====== Yusuf ”sefa” sezer

\": Çift tırnak karakteri. Çift tırnak karakterinin metin içinde kullanılmasını sağlar.

\\: Ters bölü çizgi karakteri. Ters bölü çizginin metin içinde kullanılmasını sağlar.

10 + “21” === “1021”

10 - “3” === 7

10 + “engin” === “10engin”

“engin” – “demiroğ” ======== NaN( Not a Number) sayısal bir işlem sonucunda geçerli olmayan veya tanımlanamayan bir değeri temsil eder.

3=3 reference error, 3==3 true, 3==”3” true, 3===3 true, 3===”3” false

JavaScript'te === üçlü eşitlik operatörüdür. Bu operatör, iki değerin hem değerlerini hem de veri türlerini karşılaştırır. Eğer hem değerler hem de veri türleri eşitse, true döner; aksi halde false döner.

3!==3 false, 3!==4 true

var anahtar kelimesi, eski JavaScript sürümlerinde kullanılan ve genel kapsamlı değişken tanımlamak için kullanılan bir anahtar kelimedir.

let anahtar kelimesi, blok kapsamında değişken tanımlamak için kullanılır.

const anahtar kelimesi, sabit değişken tanımlamak için kullanılır.

Prompt, kullanıcından bilgi istemek için

Number, metinsel ifadeyi sayısal ifadeye çevirmek için

If, şart geçerliyse çalışır. Else, şart geçerli değilse çalışır.

Console.log, tarayıcı konsoluna mesaj yazdırmak için kullanılır.

Function topla(a,b) ======== a,b argümandır.

Console.assert, bir hata ayıklama yöntemidir ve bir koşulun doğruluğunu kontrol eder.

console.assert(5 > 10, "Hata: 5, 10'dan büyük olmalı!");

let kullaniciAdi = 'Adem';

function mesajGoster() {

let kullaniciAdi = "Yusuf"; // yerel değişken tanımla

let mesaj = 'Merhaba, ' + kullaniciAdi; // Yusuf

alert(mesaj);

}

// buradaki fonksiyon kendi değişkenini yaratacak ve onu kullanacak.

mesajGoster();

alert( kullaniciAdi ); // Adem, değişmedi!!!, fonksiyon dışarıda bulunan değişkene erişmedi.

Yerel bir değişken dıştaki değişken ile aynı isimdedir. Dolayısıyla yerel değişken düzenlenecektir. Dıştaki değişken bundan etkilenmeyecektir.

Fonksiyon kapsamında tanımlanan değişkenler global kapsamdaki aynı isimli değişkenlerden bağımsızdır.

Eğer fonksiyon argümanına bir değer gönderilmemişse fonksiyon içerisinde bu değişken undefined olur.

Örneğin mesajGoster(gonderen,metin) fonksiyonu tek argüman ile de çağırılabilir.

mesajGoster("Mahmut");

Bu bir hata değildir. Fonksiyon eğer bu şekilde çağırılırsa, yani metin yoksa, metin == undefined varsayılır. Yukarıdaki fonksiyon çağırıldığında sonuç “Mahmut: undefined” olacaktır.

confirm bir JavaScript fonksiyonudur ve tarayıcıda kullanıcıya bir onay kutusu göstermek için kullanılır. Bu fonksiyon, kullanıcıya bir mesaj gösterir ve kullanıcının "Tamam" veya "İptal" gibi bir seçenekle yanıt vermesini bekler.

var yas = prompt('Kaç yaşındasın?', 18); İkinci parametre, giriş kutusunda önceden doldurulmuş olan varsayılan değeri belirtir.

Eğer bir fonksiyon değer döndürmüyor ise bu fonksiyon undefined döndürüyor

JavaScript'te return ifadesinden sonra bir satır boşluk bırakmaaaa!!!!

C# ve Java gibi dillerde, return ifadesinden sonra bir boşluk bırakmak veya yeni bir satıra geçmek, beklenmeyen sonuçlara neden olmaz.

Js de return

(bazı + uzun + ifade + veya + baska + birsey \* f(a) + f(b)) bu durum

return;

(bazı + uzun + ifade + veya + baska + birsey \* f(a) + f(b)); böyle algılanır.

**Fonksiyonu isimlendirme**: Fonksiyon bir fiili yerine getirdiğinden, fonksiyon isimleri yüklem olmalıdır.

Örneğin "show" fonksiyonu her zaman bir şeyleri gösterir.

Fonksiyonlar şöyle başlayabilir.

"get…" – değer döndürür,

"calc…" – bir şeyler hesaplar,

"create…" – bir şeyler yaratır,

"check…" – bir şeyleri kontrol eder ve boolean döndürür.

sendMessage(..) // mesaj gönderir

getAge(..) // yaşı döndürür

calcSum(..) // toplamı hesaplar ve geri döndürür.

createForm(..) // form oluşturur ve genelde geri döndürür.

checkPermission(..) // izni kontor eder. true/false

Fonksiyon değer döndürebilir. Eğer döndürmezse undefinedolarak tanımlanır.

Fonksiyonlar içerisinde yerel değişken kullanılması önerilir. Dış değişkenler kullanılması önerilmez.

**ÖRNEK: int yas = 18;**

**boolean yetiskinMi = (yas >= 18) ? true : false;**

**ÖRNEK2: function yasKontrolu(yas) {**

**return (yas > 18) ? true : confirm('Ebeveynlerin izin verdi mi?');**

**}**

**ÖRNEK3: function yasKontrolu(yas) {**

**return (yas > 18) || confirm('Ebeveynlerin izin verdi mi?');**

**}**

**def toplama(x, y):**

**return x + y**

**sonuc = toplama(2, 4)**

**Bu örnekte toplama() fonksiyonunun iki parametresi vardır: x ve y.**

**Bu örnekte toplama() fonksiyonu 2 ve 4 argümanlarıyla çağrılır.**

parseInt( ): String ifadenin sayısal kısmını alarak tam sayıya çevirme işlemi yapar.

parseFloat( ): parseInt fonksiyonundan farkı sayısal ifadeleri ondalıklı olarak da almak için kullanılır.

String( ): Değişken içindeki verileri string yapmak için kullanılır.

Number( ): String ifadeyi sayısal değer yapmak için kullanılır.

JavaScript fonksiyon kodları return anahtar kelimesine geldiğinde çalışmayı durdurur.

JavaScript ekrana yazdırma işlemi;

HTML nesnelerine yazdırma – innerHTML,

HTML sayfasına yazdırma – document.write(),

Uyarı pencersine yazdırma – alert(),

Tarayıcı konsoluna yazdırma – console.log()

innerHTML kullanma

JavaScript ile HTML nesnelerini seçmek için genellikle document.getElementById(ID) yöntemi kullanılır.

Seçim işleminden sonra innerHTML özelliği ile seçilen HTML nesnesinin içeriğini değiştirebiliriz.

<p id="p1"></p>

<script>

document.getElementById("p1").innerHTML = "Merhaba JavaScript";

</script>

document.write() kullanımı

JavaScript komutlarını test etmek ve sonuçları ekrana hızlı bir şekilde yazdırmak için document.write() metodu kullanılır.

<p>Burası bir paragraf</p>

<script>

document.write("Merhaba JavaScript");

</script>

document.write() metodu bir olay sonucu kullanılırsa saydaki tüm HTML elemanları silinecektir.

<p>Burası bir paragraf</p>

<button onclick="document.write('Merhaba JavaScript')">Buraya tıkla</button>

alert() kullanımı

JavaScript verilerini uyarı mesajı olarak ekran yazdırmak için alert() metodunu kullanılır.

<p>Burası bir paragraf</p>

<script>

alert("Merhaba JavaScript");

</script>

console.log() kullanımı

JavaScript Diziler, Nesneler üzerinde işlemler görüntülemek veya yazılan komutları daha performanslı hale getirmek ve ekrana yazdırmak için console.log() kullanabilirsiniz.

<p>Burası bir paragraf</p>

<script> console.log("Merhaba JavaScript"); </script>

JavaScript tanımlayıcıları küçük büyük harf duyarlıdır.

Örnekteki dunya ve Dunya iki farklı değişkendir.

Yorum satırı: // satırları /\* \*/

Sayılar tırnak içerisinde yazıldığında JavaSsript metinsel birleştirme operatörü işlemi yapacaktır.

<script>

var xSayisi = 5 + 2 + “3”;

alert(xSayisi);

</script> output: 73 tür.

Sayılar operand veya işlenen, Operatör ise işlemdir.

!!!!!!! x += y x = x + y

a == b a ve b eşit mi?

a === b a ve b değerleri ve değer türleri eşit mi?

a != b a ve b değerleri eşit değil mi?

a !== b a ve b değerleri veya değer türleri eşit mi?

typeof operatörünü kullanarak veri türlerini kontrol edebilirsiniz.

isNaN() fonksiyonu, bir değerin NaN olup olmadığını kontrol etmek için kullanılabilir.

NaN değeri, sayısal olmayan metin değerleri ve tanımsız işlemler için kullanılır. Bu yüzden,

const metinHarf = "abc";

const sayi = 123;

console.log(metinHarf == sayi); // false

console.log(metinHarf === sayi); // false

console.log(metinHarf == NaN); // true

typeof Değişken türünü verir.

instanceof Değişken türünü karşılaştırır.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!

JavaScript değişkenlerinde Sayı, Metin, Nesne veri türlerini barındırabiliriz.

var bilgiler = {ismim:"Yusuf SEZER", xSayisi:5}; kodunun nesne (object) türü olmasının sebebi, birden fazla veriyi anahtar-değer çiftleri halinde saklayan bir veri yapısı olmasındandır.

Nesne (Object) Nedir?

Nesne, birden fazla veriyi bir arada tutmak için kullanılan bir veri yapısıdır. Nesneler, anahtar-değer çiftleri koleksiyonlarıdır. Her anahtar bir değere karşılık gelir.

Bilgiler Nesnesinin Özellikleri:

bilgiler nesnesi iki anahtar-değer çifti içerir:

ismim: Anahtar. "Yusuf SEZER" değerine karşılık gelir.

xSayisi: Anahtar. 5 değerine karşılık gelir.

Bilgiler Nesnesinin Neden Nesne Türü Olduğuna Dair Açıklama:

var bilgiler ifadesi, bilgiler adında bir değişken tanımlar.

{} süslü parantezler, bir nesne oluşturmak için kullanılır.

Anahtarlar ve değerler virgülle ayrılır.

Anahtarlar metin türünden, değerler ise farklı veri türlerinden olabilir.

alert(null === undefined); // false

alert(null == undefined); // true

Deklarasyon: Bir değişkenin veya fonksiyonun adını, türünü ve diğer özelliklerini tanımlama işlemidir.

Hoisting, JavaScript'te değişkenlerin ve fonksiyonların deklarasyonlarının kodun en başına taşınması anlamına gelir. Bu, değişkenlerin veya fonksiyonların kullanılmadan önce tanımlanmasına gerek kalmadan JavaScipt ile değişken oluşturmadan değişkene değer atanırsa hoisting sayesinde değişken GLOBAL olur. Değişken oluşturmadan değer atamak tavsiye edilmez.

JavaScript global değişkenler window nesnesinde tutulur.

Window nesnesi: JavaScript'te tarayıcı penceresini temsil eden bir nesnedir.

Window Nesnesinin Özellikleri:

location: Tarayıcı penceresinin URL'sini ve diğer konum bilgilerini içeren bir nesnedir.

document: Tarayıcı penceresinde görüntülenen HTML belgesini temsil eden bir nesnedir.

history: Tarayıcı penceresinin geçmişini içeren bir nesnedir.

navigator: Tarayıcı hakkında bilgi içeren bir nesnedir.

screen: Tarayıcı penceresinin ekranı hakkında bilgi içeren bir nesnedir.

Değişken yaşam süresi

JavaScript değişkenin yaşam süresi değişken oluşturulduğunda başlar.

Yerel değişkenler fonksiyon işlemini tamamladığında yaşam süresi biter ve silinir.

Genel değişkenlerin yaşam süresi web sayfası kapatıldığında biter.

**Olaylar:**

HTML nesneleri üzerinde oluşan her eylem bir olaydır.

JavaScript HTML nesneleri üzerinde oluşan olay sonucunda işlem yapabilir.

onchange HTML öğesi değiştirildiğinde tetiklenir.

onclick Kullanıcı HTML nesnesine tıkladığında tetiklenir.

onmouseover Kullanıcı HTML nesnesinin üzerine geldiğinde tetiklenir.

onmouseout Kullanıcı HTML nesnesinin üzerinden ayrılınca tetiklenir.

onkeydown Kullanıcı klavye tuşuna basınca tetiklenir.

onload Tarayıcı sayfayı yüklemeyi bitirdiğinde tetiklenir.

**Metin uzunluğu**

Metin uzunluğu için length özelliği kullanılır. alert(ismim.length);

**Metin içinde arama**

Metin içinde arama yapmak için indexOf fonksiyonu kullanılır. alert(metin.indexOf("Yusuf"));

Aranan değer bulunmazsa indexOf ve lastIndexOf fonksiyonları -1 değeri verir.

Metin içinde arama yapmak için search fonksiyonu da kullanılır.

**Metin parçalama**

Metin parçalamak için 3 fonksiyon kullanılır.

slice(başlangıç, bitiş);

substring(başlangıç, bitiş);

substr(başlangıç, uzunluk);

Tek fark substring fonksiyonu negatif değer kabul etmez.

İlk parametre negatif olursa başlangıç pozisyonu değerin sonundan başlar.

İkinci parametre uzunluk olduğundan negatif olamaz.

**Metin değiştirme**

Metin içindeki değeri başka bir değer ile değiştirmek için replace fonksiyonu kullanılır

var metin = "Merhaba ben Sefa SEZER";

alert(metin.replace("Sefa", "Yusuf"));

**Metni büyük ve küçük harfe çevirme**

Metin içindeki harfleri büyük harfe çevirmek için toUpperCase fonksiyonu kullanılır.

Metin içindeki harfleri küçük harfe çevirmek için toLowerCase fonksiyonu kullanılır.

**Metin birleştirme**

İki veya daha fazla metni birleştirme için concat fonksiyonu kullanılır.

alert(metin.concat(" ",ismim));

JavaScript concat fonksiyonu yerine birleştirme operatörü (+) kullanılabilir.

alert(metin + " " + ismim);

**Metinden karakter alma**

charAt fonksiyonu metin veri türündeki parametre ile belirtilmiş karakteri verir.

alert(metin.charAt(6));

**Metni diziye dönüştürme**

split Bir metni belirli bir ayraç karakterine göre parçalara ayırır ve her bir parçayı bir dizi öğesi olarak döndürür.

var str = "Merhaba, dünya!";

var arr = str.split(" ");

console.log(arr); // ["Merhaba,", "dünya!"]

**Sayı sistemleri**

JavaScript sayı değeri 0x ile başlıyorsa sayıyı onaltılık sayı sistemi olarak tanımlar.

JavaScript sayıları varsayılan olarak onluk sayı sistemi olarak tanımlar.

Sayıları onaltılık, sekizlik ve ikilik sayı sistemine dönüştürmek için toString fonksiyonu kullanılır.

var x = 1453;

alert(x.toString(16)); // 5ad

alert(x.toString(8)); // 2655

alert(x.toString(2)); // 10110101101

**NaN – Not a Number**

Bir değişkenin sayı olup olmadığını NaN anahtar kelimesiyle öğrenebiliriz.

Sayısal bir değer sayısal olmayan bir değerle işlem yapılırsa JavaScript NaN sonucunu verir.

Metin veri türü içindeki değer sayı olursa sonuç sayı değeri olacaktır.

Değişkenin sayı olup olmadığını bulmak için isNaN fonksiyonu kullanılır.

**Değişkenleri sayıya çevirme**

Sayısal olmayan değerleri sayıya çevirmek için 3 fonksiyon kullanılır.

Number() fonksiyonu

parseInt() fonksiyonu

parseFloat() fonksiyonu

**Number**

Number fonksiyonu aldığı değeri sayı türüne çevirmek için kullanılır.

**parseInt**

parseInt fonksiyonu aldığı değeri sayı türüne çevirmek için kullanılır.

Aldığı değeri sayı türüne çeviremezse NaN değerini verir.

**parseFloat**

parseFloat fonksiyonu aldığı değeri sayı türüne çevirmek için kullanılır.

Sayılar arasında boşluk varsa ilk sayıyı verir.

Aldığı değeri sayı türüne çeviremezse NaN değerini verir.

Math.round

Math.round fonksiyonu parametre olarak verilen ondalıklı sayıya en yakında tam sayıyı değerini verir.

Math.pow()

Math.sqrt

Math.abs, mutlak değer

Math.ceil()

Math.ceil fonksiyonu parametre olarak verilen ondalıklı sayıyı yukarı yuvarlar.

Math.min()

Math.min fonksiyonu parametre olarak verilen sayılar arasında en küçük sayıyı verir.

alert(Math.min(0, 30, 359, 100, -1, -100)); // -100

Math.random()

Math.random fonksiyonu 0 ile 1 arasında rastgele sayı üretir.

**Tarih bilgisini almak**

getDate() Ayın gününü verir (1-31)

getDay() Haftanın gününü verir (0-6)

getFullYear() Yılı dört basamak olarak verir (yyyy)

getHours() Saati bilgisini verir (0-23)

getMilliseconds() Milisayine bilgisini verir (0-999)

getMinutes() Dakika bilgisin verir (0-59)

getMonth() Ay bilgisini verir (0-11)

getSeconds() Saniye bilgisini verir (0-59)

getTime() Tarihi 1 Ocak , 1970’dan itibaren milisaniye olarak verir

**Diziler**

Bir değişkende birden fazla değer tutabilen değişken türüdür.

JavaScript dizi oluşturmak için;

var liste = [];

var liste = new Array();

Ancak new Array() kullanımı pek tercih edilmez.

!!!!! Dizilerin türü objecttir. Nesnelerin özel bir türüdür.

**Diziler üzerinde gezmek**

JavaScript diziler üzerinde sırayla işlem yapmak veya okumak için genellikle for döngüsü kullanılır.

<script>

var liste, i, listeUzunluk;

liste = ["Elma", "Armut", "Portakal"];

listeUzunluk = liste.length;

for(i = 0; i < listeUzunluk; i++) {

alert(liste[i]);

}

</script>

**Diziye elaman ekleme**

Diziye yeni bir eleman eklemek için push fonksiyonu kullanılabilir.

<script>

var liste = ["Elma", "Armut", "Portakal"];

liste.push("Muz");

alert(liste);

</script>

**JavaScript Dizi Fonksiyonları**

JavaScript dizi fonksiyonları diziler üzerinde işlem yapmamızı sağlar.

toString

JavaScript toString fonksiyonu dizi elamanlarını virgülle ayrılmış metin veri türüne dönüştürür.

<script>

var liste = ["Elma", "Armut", "Portakal"];

alert(liste.toString()); </script>

join

Dizi elemanlarını birleştirmek için join fonksiyonu kullanılır.

pop

Dizinin son elemanını kaldırmak için pop() fonksiyonu kullanılır.

push

Dizinin sonuna yeni eleman eklemek için push fonksiyonu kullanılır.

shift

Dizinin ilk elemanını kaldırmak için shift fonksiyonu kullanılır. Diğer elemanları bir öne kaydırır.

unshift

Dizinin önüne yeni eleman eklemek için unshift fonksiyonu kullanılır.

Diğer elemanları bir sona kaydırır.

Dizi elemanını silmek

JavaScript delete anahtar kelimesi ile dizi elemanı silinebilir.

delete anahtar kelimesini kullanmak dizi arasında boşlukların oluşmasına neden olur.

Diziye yeni eleman eklemek

splice() fonksiyonu kullanılabilir.

<script>

var liste = ["Elma", "Armut", "Portakal"];

alert(liste);

liste.splice(1, 0, "Muz", "Kivi");

alert(liste);

concat

Dizileri birleştirerek yeni dizi oluşturmak için concat fonksiyonu kullanılır.

slice

Diziden belli elemanları almak için slice fonksiyonu kullanılır.

Alfabetik sıralama

Dizi elamanlarını alfabetik olarak sıralamak için sort fonksiyonu kullanılır.

sort fonksiyonu ile sıraladığımızda metin veri türü olarak sıralayacak

<script>

var liste = ["Elma", "Armut", "Portakal", "Muz", "Kivi", "Karpuz", "Çilek"];

liste.sort();

alert(liste);

Ters sıralama

Dizi elemanlarını tersine çevirmek için reverse fonksiyonu kullanılır.

JavaScript Türkçe sıralama

JavaScript ASCII karakterleri dışındaki karakterleri sıralamaz.

JavaScript ile Türkçe sıralama yapmak için localeCompare fonksiyonu kullanılır.

switch case

Birden fazla değeri karşılaştırmak için switch case yapısı kullanılabilir.

JavaScript for döngüsü koşul

Koşul alanındaki karşılaştırma sonucunda değer true olursa döngü devam eder false olursa döngü biter.

Koşul alanına herhangi bir şey yazılmadan da kullanılabilir.

for( ; ; i++) {

alert(liste[i]);

if(i == 6) break;

}

Komut 2 alanı yazılmadan kullanılabilir.

for(i = 0; i < 11;) {

sonuc += i;

i += 2;

}

for in döngüsü foreach

for in döngüsü genellikle eleman sayısını bilmediğimiz diziler veya elemanlarını bilmediğimiz nesnelerde kullanılır.

for(i in liste) {

alert(liste[i]);

}

continue

continue anahtar kelimesi döngü adımını atlamak için kullanılır.

JavaScript değişken tipleri

JavaScript’te değer saklanabilen 5 değişken türü vardır.

string

number

boolean

object

function

JavaScript’te 3 farklı nesne türü vardır.

Object

Date

Array

JavaScript’te değer içermeyen 2 değişken türü vardır.

null

undefined

**Düzenli ifadeler**

Metin veri türü içerisinde arama yapmak için düzenli ifadeler kullanılır.

Düzenli ifadeler bir karakterden oluşacağı gibi daha karmaşık karakterlerden oluşabilir.

Düzenli ifadeler metin veri türü içinde arama yapma ve değiştirme işlemleri için kullanılır.

JavaScript regex kullanımı

/ifade/nitelik;

Örnek

var aranan = /yusuf/i;

/yusuf/i düzenli bir ifadedir.

yusuf aranacak kelimedir.

i küçük büyük harfe duyarsız arama yapılacağını ifade eder.

Regular expression kullanımı

Metin içinde arama yapmak için search fonksiyonu kullanılır.

Metin içindeki değeri başka bir değer ile değiştirmek için replace fonksiyonu kullanılır.

var metin = "Merhaba ben YuSuF";

var sonuc = metin.search(/yusuf/i);

alert(sonuc); output====12

replace

Örnekte düzenli ifade yusuf kelimesini küçük büyük harfe duyarsız olarak Sefa kelimesi ile değiştirir.

var metin = "Merhaba ben YuSuF";

var sonuc = metin.replace(/yusuf/i, "Sefa"); alert(sonuc); output====Merhaba ben Sefa

JavaScript düzenli ifade nitelikleri

Nitelik Açıklama

i Küçük büyük harf duyarsız arama yapar.

g Metin veri türü içerisinde aranan tüm kelimeler üzerinde işlem yapar.

m Çok satırlı arama yapar.

JavaScript düzenli ifade desenleri

Köşeli parantez içerisinde desenler kullanarak belirli karakter araması yapılır.

Desen Açıklama

[abc] Köşeli parantezler arasındaki karakterlerden herhangi birini bul

[0-9] Parantezler arasındaki herhangi bir rakamı bul

(x|y) Ayraç ile ayrılmış seçeneklerden herhangi birini bul

Meta karakterler özel bir anlam taşıyan karakterlerdir.

Meta karakter Açıklama

\d Sayıları bul

\s Boşluk karakterini bul

\b Kelimenin başında veya sonunda bul

\uxxxx Onaltılık sayı xxxx ile belirtilen Unicode karakterini bul

Niceleyiciler arama niceliğini belirtir.

Nicelik Açıklama

n+ En az bir n içeren herhangi bir kelime bul.

n\* Sıfır veya daha fazla n içeren herhangi bir kelime bul.

n? Sıfır veya bir kez n‘yi içeren herhangi bir kelime bul.

test

test fonksiyonu düzenli ifade ile arama yapar.

var ifade = /yusuf/i

var sonuc = ifade.test("Merhaba ben YuSuF");

alert(sonuc); Aranan ifade varsa true yoksa false değerini verir.

exec

JavaScript exec fonksiyonu düzenli ifade ile arama yapar.

var ifade = /yusuf/i

var sonuc = ifade.exec("Merhaba ben YuSuF");

alert(sonuc);

Aranan ifade varsa bulunan kelimeyi yoksa null değerini verir.

**Hataları yakalamak ve hataları yönetmek** için kullanılan try catch finally anahtar kelimelerinin kullanımı, JavaScript hataları ile ilgili bilgiler yer alıyor.

JavaScript try anahtar kelimesi kod bloğundaki kodları çalışma zamanında test etmek için kullanılır.

JavaScript catch anahtar kelimesi çalışma zaman hatası sonucu oluşan hataları ekrana yazdırmak için kullanılır.

JavaScript throw anahtar kelimesi özel hata oluşturmayı sağlar.

JavaScript finally anahtar kelimesi hata oluşması veya oluşmaması durumunda çalışacak kodları yazdırmak için kullanılır.

Try, test için. Hataları tespit etmek için. Catch, eğer hata varsa bunu ekrana yazdrmak için.

Try:

Bu blok, hataya neden olabilecek kodları içerir.

Bir hata oluşmazsa, kod çalışmaya devam eder.

Catch:

Bu blok, try bloğunda bir hata oluşursa çalıştırılır.

Hata hakkında bilgi içeren bir hata nesnesi alır.

Hata mesajını ekrana yazdırmak veya hatayı işlemek için bu nesneyi kullanabilirsiniz.

throw

JavaScript kodları çalıştırılırken hata meydana geldiğinde JavaScript durur ve throw hata mesajı üretir.

JavaScript hata oluştuğunda name ve message özelliklerine sahip Error nesnesi oluşturacaktır.

JavaScript throw anahtar kelimesi ile özel hata oluşturabiliriz.

JavaScript hata mesajları metin, sayı, mantık ve nesne türünden olabilir.

finally

JavaScript finally anahtar kelimesi sonuç ne olursa olsun çalıştırılacak kodların yazılmasını sağlar.

Error nesnesi

JavaScript hata oluştuğunda hata bilgisini name ve message olarak saklayan Error nesnesini kullanır.

name Hata adını saklar

message Hata mesajını saklar

JavaScript hataları

JavaScript içerisinde çeşitli hata türleri mevcuttur.

RangeError Sayı sınırı hatası

ReferenceError Referans hatası

SyntaxError Sözdizimi hatası

TypeError Tür hatası

URIError encodeURI() fonksiyonu hatası, geçersiz karakter kullanımında

**JavaScript hata ayıklama**

tarayıcılarda JavaScript hata ayıklayıcısı mevcuttur.

console.log

var a = 2;

var b = 3;

var c = a + b;

console.log(c);

debugger

Hata ayıklama penceresi açıksa, aşağıdaki kod dördüncü satıra geçmeden önce durur.

var a = 2;

var b = 3;

debugger;

var c = a + b;

console.log(c);

**use strict** JavaScript'te "strict mode" (katı mod) adı verilen bir modu etkinleştirmek için kullanılan bir direktiftir. Bu mod, kodun daha güvenli, daha öngörülebilir ve daha az hata içeren bir şekilde yazılmasını sağlar.

"use strict";

a = 2;

b = 3;

c = a + b;

alert(c); // Değişken oluşturulmadan kullanıldığından hata verir.

Katı kural tanımı fonksiyon içerisinde yapılırsa sadece fonksiyon içini kapsar.

m = 45;

alert(m);

hesapla();

function hesapla() {

"use strict";

a = 2;

b = 3;

c = a + b;

alert(c); // Değişken oluşturulmadan kullanıldığından hata verir.

}

Değişken tanımlamadığımız için fonksiyon içinde a,b,c değeri için hata verir.!!!!

**Neden use strict?**

JavaScript katı kural tanımı güvenli kod yazmak için kullanılır.

Katı kural tanımı daha önceki hatalı kullanımın önüne geçer.

Örnek olarak JavaScript’te değişken adını yazarak bir değişken kullanabiliriz.

Ancak katı kural tanımı ile değişken oluşturmadan değişken kullanmak hata verecektir.

Değişken oluşturulmadan kullanmak hata verir.

Nesne veri türü oluşturulmadan kullanmak hata verir.

Değişken ve nesnenin silinmesine izin verilmez.

Fonksiyon silinmesine izin verilmez.

Fonksiyona aynı parametrenin iki defa yazılmasına izin verilmez.

Sekizli sayısal değerlere izin verilmez.

Kaçış karakterlerine izin verilmez. (özel anlamlara sahip olan karakterler)

Değişken ismi olarak eval kullanılmasına izin verilmez.

Değişken ismi olarak arguments kullanılmasına izin verilmez.

JavaScript with kullanımına izin verilmez. (bir kod bloğunu çalıştırmak için kullanılır.) kod okunabilirliği azalır ve kapsam sorunlarına yol açabilir.

// Önerilmeyen kullanım:

const person = {

name: "Ahmet"

};

with (person) {

console.log(name);

}

// Daha güvenli kullanım:

console.log(person.name + " " + person.age);

JavaScript eval fonksiyonu ile oluşturulmuş değişkenlerin kullanımına izin verilmez. (Bir metin stringini JavaScript kodu olarak çalıştırmak için kullanılır. Bu, dinamik kod oluşturmak için kullanılabilir.)

// Önerilmeyen kullanım:

const code = "let x = 10; console.log(x);";

eval(code);

// Daha güvenli alternatif:

function calculate(value) {

let x = value;

console.log(x); }

Use scrictin izin vermediği kelimeler:

Katı kural tanımlanırsa özel JavaScript anahtar kelimelerin kullanımına izin verilmez.

İmplements: Bir sınıfın, bir arayüzün soyut metotlarını uyguladığını belirtmek için kullanılır.

İnterface: Bir sınıfın sahip olması gereken metotların bir tanımını içeren bir şablondur.

let: Bir değişkeni blok kapsamında tanımlamak

package: birden fazla sınıfı ve arayüzü bir araya getirmek için kullanılır.

Private: o sınıfın içinde erişilebilir

protected: o sınıfın ve alt sınıflarının içinde erişilebilir

public: tüm sınıflardan erişilebilir

static: üyeler sınıftan bağımsız olarak kullanılabileceği

yield: Jeneratör fonksiyonlarında kullanılır. Bir jeneratör fonksiyonu, bir dizi değeri döngüsel olarak döndürmek için tasarlanmıştır. yield anahtar kelimesi, jeneratör fonksiyonundan bir değeri döndürmek için kullanılır.

function\* sayilar() {

yield 1;

yield 2;

yield 3;

}

const iterator = sayilar();

console.log(iterator.next().value); // 1

console.log(iterator.next().value); // 2

console.log(iterator.next().value); // 3

**JS de karşılaştırma ve atama operatörü:**

**== Eşitse**

**=== Hem değeri hem de türü eşitse**

**!= Eşit değilse**

**= Değer atama**

**+= Toplama ve atama**

**-= Çıkarma ve atama**

**\*= Çarpma ve atama**

**/= Bölme ve atama**

**%= Mod alma ve atama**

**DİKKAT!!!**

1-)Global değişen kullanmayın!

2-)Değişkenleri en başta oluşturun! Değişkenleri komut veya fonksiyonun en başında oluşturun. Yazılan kodlar daha okunabilir olacak ve kod karmaşıklığını en aza indirecektir.

3-)Değişkenleri oluşturduktan sonra varsayılan değer tanımlayın. Değişkenler daha güvenilir ve tanımsız olmayacaktır.

// Değişkenler oluşturuldu ve tanımlandı

var adi = "", soyAdi = "", fiyat = 0, indirim = 0, urunFiyat = 0;

4-)Number, String ve Boolean veri türünü nesne olarak tanımlamayın.

Sayı, metin ve mantık veri türlerini daima temel veri türü olarak tanımlayın ve kullanın.

Sayı, metin ve mantık veri türlerini nesne olarak kullanmak kod yavaşlığına ve diğer yan etkilere neden olur.

5-)JavaScript new anahtar kelimesi kullanmayın.

new Object() yerine {} kullanın.

new String() yerine “” kullanın.

new Number() yerine 0 kullanın.

new Boolean() yerine false kullanın.

new Array() yerine [] kullanın.

new RegExp() yerine /()/ kullanın.

new Function() yerine function (){} kullanın.

6-)Otomatik tip dönüşümlerine dikkat edin

JavaScript otomatik tip dönüşümleri ile sayılar yanlışlıkla metin veri türüne dönüştürülebilir ve istenmeyen sonuçlar verebilir.

7-)Karşılatırmada === kullanın

İki farklı değeri == ile karşılaştırıldığında karşılaştırma yapılmadan önce veri türü dönüşümü yapılır.

Ancak === ile karşılaştırma yapıldığında veri türü dönüşümü yapılmadan değerler karşılaştırılır.

8-)Varsayılan parametre değeri

Fonksiyon parametreleri eksik gönderilirse eksik parametre tanımsız undefined olur.

Tanımsız değişken fonksiyonu çalışmaz hale getirebilir.

Fonksiyon parametrelerine varsayılan değer atayarak bu sorun çözülebilir.

9-)switch deyimine varsayılan ekleyin

switch deyimine default deyimi gerek olmasa bile ekleyin.

10-)eval fonksiyonunu kullanmayın

JavaScript eval() fonksiyonu parametre olarak yazılan kodu çalıştırmak için kullanılır.

Kodları isteğe bağlı çalıştırdığından, güvenlik sorunları oluşturabilir.

11-)let ile tanımlanan değişkenler blok kapsamlıdır, global değildir. !!!!!

Tür (Type): Bir değişkenin değerinin hangi veri kategorisine ait olduğunu belirtir. JavaScript'te temel veri türleri number, string, boolean, null, undefined ve symbol'dur.

let ve var: Değişkenleri tanımlamak için kullanılan anahtar kelimelerdir.

12-)let, blok kapsamlı global değildir. Daha sık tercih edilir.

Var, fonksiyon kapsamlı global olabilir(tercih edilmez). Daha az tercih edilir.

13-) satır değerin ortasından bölünemez.

var a = "Merhaba

JavaScript";

alert(a);

Satırı değerin ortasında bölmek için ters eğik çizgi kullanılır.

var a = "Merhaba \

JavaScript";

alert(a);

14-) Geri değer döndürmeyi birden fazla satıra bölmeyin. Return

15-) JavaScript with deyimini kullanmayın! with, eval

**JavaScript Gizle Göster**

CSS display özelliği ile HTML nesnelerinin görünüm ayarını değiştirebilir veya gizleyebiliriz.

HTML nesnesine display özelliği verilmemişse none ile nesneyi gizliyor, none değeri verilmişse none değerini kaldırıyoruz.

<button onclick="gizleGoster('sonuc');">Gizle / Göster</button>

<p id="sonuc">Gizlenip gösterilen nesne</p>

<script> function gizleGoster(ID) {

var secilenID = document.getElementById(ID);

if (secilenID.style.display == "none") {

secilenID.style.display = "";

} else {

secilenID.style.display = "none";

}

} </script>

**JavaScript Yönlendirme**

Web sayfamızı başka bir alan adına veya başka bir sayfaya taşıdığımızda mevcut sayfayı JavaScript yönlendirme kullanarak yönlendirebiliriz.

<script> window.location.href = "http://www.yusufsezer.com.tr"; </script>

**JavaScript rastgele tam sayı üretmek**

JavaScript Math.random() ile üretilen rastgele ondalıklı sayıyı Math.floor() ile ondalıksız tamsayıya çevirebiliriz.

<script> alert(Math.floor(Math.random() \* 10)); // 0 ve 9 arasında sayı üretir. </script>

Form seçim işlemi document nesnesi forms özelliği ile yapılabilir.

**Document nesnesi özellikleri şunlardır:**

document.URL: HTML belgesinin URL'sini verir.

document.title: HTML belgesinin başlığını verir.

document.documentElement: HTML belgesinin <html> elementini verir.

document.body: HTML belgesinin <body> elementini verir.

document.head: HTML belgesinin <head> elementini verir.

document.forms: HTML belgesindeki tüm formları içeren bir NodeList döner.

document.images: HTML belgesindeki tüm resimleri içeren bir NodeList döner.

document.links: HTML belgesindeki tüm bağlantıları içeren bir NodeList döner.

document.scripts: HTML belgesindeki tüm komut dosyalarını içeren bir NodeList döner.

document.getElementById(id): id özelliğine sahip bir HTML elementini döner.

document.getElementsByClassName(className): className özelliğine sahip tüm HTML elementlerini içeren bir NodeList döner.

document.getElementsByTagName(tagName): tagName etiketine sahip tüm HTML elementlerini içeren bir NodeList döner.

document.createElement(tagName): Belirtilen tagName etiketine sahip yeni bir HTML elementini oluşturur.

document.createTextNode(text): Belirtilen metni içeren yeni bir TextNode oluşturur.

innerHTML: içerik değerini atamak için kullanılır.

**HTML Sınırlamaları:**

HTML, verileri yapılandırmak için yeterince uygun değildir. Veri yapılandırmak demek, verileri organize etmek, düzenlemek anlamına gelir. Verileri daha kolay bulmak, anlamak ve kullanmak için.

Verileri yapılandırmanın birkaç yolu vardır:

Veritabanları: Verileri tablolar halinde saklamak için veritabanları kullanılabilir. Bu, verileri kategorilere ve ilişkilere göre organize etmenin bir yoludur.

XML(eXtensible Markup Languge): XML, verileri etiketler ve öznitelikler kullanarak yapılandırmak için kullanılan bir işaretleme dilidir. Bu, verileri platformdan bağımsız bir şekilde saklamak ve taşımak için bir yoludur.

JSON(JavaScript Object Notation): JSON, verileri anahtar-değer çiftleri kullanarak yapılandırmak için kullanılan bir veri değişim formatıdır. Bu, verileri JavaScript ile kolayca işlemek için bir yoludur.

CSV(Comma-Separated Values): CSV, verileri virgülle ayrılmış değerler kullanarak yapılandırmak için kullanılan bir dosya formatıdır. Bu, verileri elektronik tablolar gibi diğer programlarda kolayca açmak için bir yoludur.

**XML (Genişletilebilir İşaretleme Dili)**

XML eXtensible Markup Languge (Genişletilebilir İşaretleme Dili) kısaltmasıdır.

XML, etiket tabanlı bir dildir.

XML verileri hem insan hem de makineler tarafından anlaşılır şekilde depolamak ve farklı platformlar arasında taşımak için W3C tarafından standart olarak belirlenmiş, etiket tabanlı bir dildir.

**Ne yapılır?**

XML verileri saklamak için bir yapı oluşturulabilir.

Verileri farklı platformlar arası taşımak için kullanılabilir.

Web sayfaları arasında veri paylaşımı için kullanılabilir.

RSS, Feed, Ajax ve Web servisleri oluşturulabilir.

**Niçin kullanılır?**

XML, verileri yapılandırmak ve taşımak için kullanılır.

**Özellikleri ve kuralları**

XML dosyalarının başında XML tanımlayıcısının kullanılması faydalı olacaktır.

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

Tüm etiketler bir kök etikete sahiptir.

Etiketlerin kapatılması gerekir.

Etiketlerin içeriği boşsa

<etiket />

şeklinde kapatılmalıdır.

Etiketlerin özellikleri boş bırakılamaz.

Etiketler karışık kapatılamaz.

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

<kisiler>

<kisi>

<sira>1</sira>

<adi>Yusuf</adi>

<soyadi>SEZER</soyadi>

</kisi>

<kisi>

<sira>2</sira>

<adi>Ramazan</adi>

<soyadi>SEZER</soyadi>

</kisi>

<kisi>

<sira>3</sira>

<adi>Sinan</adi>

<soyadi>SEZER</soyadi>

</kisi>

<kisi>

<sira>4</sira>

<adi>Mehmet</adi>

<soyadi>SEZER</soyadi>

</kisi>

</kisiler>

**AJAX nedir?**

AJAX Asynchronous JavaScript And XML yani Eşzamanlı JavaScript ve XML kısaltmasıdır.

JavaScript AJAX teknolojisi web sayfalarını yenilemeden arka planda veri gönderme ve veri alma işlemleri için kullanılır.

AJAX kullanımı

AJAX istekleri XMLHttpRequest nesnesi ile yapılır.

<script>

'use strict';

var xhr = new XMLHttpRequest();

</script>

XMLHttpRequest nesnesi çeşitli metot, olay ve özelliğe sahiptir.

XMLHttpRequest nesnesine ait open metodu AJAX isteği açmak için kullanılır.

XMLHttpRequest.open(method, url, async, user, password);

Metota ilk iki parametrenin eklenmesi yeterli olacaktır.

<script> 'use strict';

var xhr = new XMLHttpRequest();

xhr.open('GET', 'url-adres'); </script>

AJAX isteği sırasında XMLHttpRequest nesnesine ait olaylar(onloadstart, onloadend, onerror, …) tetiklenir.

AJAX işlemleri durum her değiştiğinde tetiklenen onreadystatechange olayı içinde yapılır.

AJAX işlem durumu readyState özelliği ile kontrol edilir.

İstek başlatılmadığında readyState özelliği 0, açıldığında 1, istek gönderildiğinde 2, yüklendiğinde 3 ve tamamlandığında 4 değerini alır.

İşlemleri readyState değeri 4 ve HTTP durum kodunun 200 olması ile gerçekleştirmek olası hataları önleyecektir.

<script> 'use strict';

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

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

// işlemler

}

}xhr.open('GET', 'url-adres'); </script>

İstek sonucunda dönen değerler response, responseText ve responseXML ile alınır.

<script> 'use strict';

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

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

console.log(this.response);

}

}

xhr.open('GET', 'url-adres'); </script>

AJAX isteği başlatılmadığından herhangi bir dönüş olmayacaktır.

AJAX isteği send metodu ile başlatılır.

<script> 'use strict';

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

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

console.log(this.response);

}

}

xhr.open('GET', 'url-adres');

xhr.send(); </script>

Metot ayrıca setRequestHeader metodu ile birlikte POST metoduna göre veri göndermek içinde kullanılır.

<script> 'use strict';

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

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

console.log(this.response);

}

}

xhr.open('POST', 'url-adres');

xhttp.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');

xhr.send(); </script>

NOT: İstek sırasında adresin yanlış yazılması veya tarayıcının isteği engellemesinden hata oluşabilir.

AJAX işlemleri benzerlik gösterdiğinden kod tekrarını azaltmak için AJAX işlemlerini fonksiyon haline getirmek tekrar kullanım için faydalı olacaktır.

<script> 'use strict';

function ysAjax(url, method, data, callback) {

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

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

if (typeof callback == 'function') {

callback(this);

}

}

}

xhr.open(method, url);

if (method == 'POST' && !(data instanceof FormData)) {

xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');

}

xhr.send(data);

} </script>

Ajax ile dosya okuma örneği aşağıda yer almaktadır.

<script>

'use strict';

ysAjax('metin.txt', 'GET', null, yazdir);

function yazdir (sonuc) {

console.log(sonuc.responseText);

}

</script>

**Javascriptte Nesneler:**

JavaScript değişkenleri tek değer saklar.

Ancak nesneler birden fazla değer saklayabilir.

Nesne değerleri ad:değer çifti olarak yazılır. (ad ve değer iki nokta (:) ile ayrılır)

<script> var kisi = {adi:"Yusuf", soyAdi:"SEZER", yas:75};

console.log(kisi.adi + " " + kisi.soyAdi); </script>

**Tüm nesne özelliklerine erişim**

JavaScript nesnelerine ait tüm özeliklere erişmek için for in döngüsü kullanılabilir.

<script>

var kisi = {

adi:"Yusuf",

soyAdi:"SEZER",

yas:75

};

for (x in kisi) {

console.log(x + " " + kisi[x]);

}

</script>

**Nesneye yeni özellik ekleme**

Nesneye yeni özellik aşağıdaki gibi eklenebilir.

<script>

var kisi = {

adi:"Yusuf",

soyAdi:"SEZER",

yas:75

};

kisi.gozRengi = "Siyah";

for (x in kisi) {

console.log(x + " " + kisi[x]);

}

</script>

**Nesne özelliğini silme**

JavaScript delete anahtar kelimesi nesne özelliğini ve değerini siler.

<script>

var kisi = {

adi:"Yusuf",

soyAdi:"SEZER",

yas:75

};

delete kisi.yas;

console.log("Merhaba ben " + kisi.adi + " " + kisi.soyAdi + " yaşım : " + kisi.yas);

</script>

**Javascript Nesne Yönelimli Programlama**

Temel veri türleri(string, number, bool, null, undefined) haricinde tüm JavaScript değerleri bir nesnedir.

Nesneler

JavaScript değişkenleri tek değer saklar.

<script> var kisi = "Yusuf SEZER"

console.log(kisi); </script>

JavaScript nesneleri de değişkendir.

Ancak nesneler birden fazla değer saklayabilir.

Nesne değerleri ad:değer çifti olarak yazılır. (ad ve değer iki nokta (:) ile ayrılır)

<script> var kisi = {adi:"Yusuf", soyAdi:"SEZER", yas:75};

console.log(kisi.adi + " " + kisi.soyAdi); </script>

Özellik nitelikleri

JavaScript nesnelerine çeşitli kısıt getirmek için enumerable, configurable ve writable nitelikleri kullanılır.

Bu nitelikler özelliğin döngü ile okunabilir, delete anahtar kelimesi ile silinebilir ve yeni değer atayarak yazılabilir olup olmadığını belirtir.

<script> var kisi = {};

Object.defineProperty(kisi, "adi", {

enumerable: false,

configurable: false,

writable: false,

value: "Yusuf"

});

delete kisi.adi; // configurable false silinemez

kisi.adi= "Cenk"; // writable false değiştirilemez

for (x in kisi) {

console.log(kisi[x]); // enumerable false döngü içinde kullanılamaz

} </script>

!!! Nesneye metot eklenebilir.

Nesneye metot ekleme Nesneye metot ekleme aşağıdaki gibidir.

metotAdi : function () { kod bloğu }

Nesne metotlarına erişim Nesne içerisindeki bir metoda erişim aşağıdaki gibidir.

nesneAdi.metotAdi();

Nesne metoduna erişim örneği aşağıda yer almaktadır.

<script> var kisi = {

adi:"Yusuf",

soyAdi:"SEZER",

yas:75,

adiSoyad: function() {

return this.adi + " " + this.soyAdi;

}

};

console.log(kisi.adiSoyad()); </script>

Parantezler () unutulursa JavaScript fonksiyon tanımını döndürür.

Nesne kurucusuna metot ekleme

Nesne kurucusuna metot ekleme kurucu fonksiyon içerisinde yapılır.

<script> function Kisi(a, s, y) {

this.adi = a;

this.soyAdi = s;

this.yas = y;

this.adiDegistir = function (yeniAd) {

this.adi = yeniAd;

}

}

var mehmet = new Kisi("Mehmet", "SEZER", 45);

var yusuf = new Kisi("Cenk", "SEZER", 75);

console.log(yusuf.adi); // Cenk

yusuf.adiDegistir("Yusuf");

console.log(yusuf.adi); // Yusuf </script>

kisi nesnesi adiDegistir metodu nesnenin sahip olduğu adi özelliğini değiştirmek için kullanır.

Yukarıdaki örnekte; mehmet ve yusuf nesneleri yer almaktadır.

Metot hangi nesnenin değişeceğini metot içerisindeki this anahtar kelimesi ile bilir.

Nesne oluşturma

Nesneler çeşitli şekillerde oluşturulabilir. 4 çeşit

1-) Sabit değer (literal) ile nesne oluşturma ve tanımlama. <script>

var kisi = {

adi:"Yusuf",

soyAdi:"SEZER",

yas:75

};

console.log("Merhaba ben " + kisi.adi + " " + kisi.soyAdi + " yaşım : " + kisi.yas); </script>

2-) new anahtar kelimesi ile nesne oluşturma ve tanımlama. <script>

var kisi = new Object();

kisi.adi = "Yusuf";

kisi.soyAdi = "SEZER";

kisi.yas = 75;

console.log("Merhaba ben " + kisi.adi + " " + kisi.soyAdi + " yaşım : " + kisi.yas); </script>

3-) Nesne kurucusu(constructor) ile nesne oluşturma ve tanımlama. <script>

function Kisi(a, s, y) {

this.adi = a;

this.soyAdi = s;

this.yas = y;

}

var yusuf = new Kisi("Yusuf", "SEZER", 75);

var mehmet = new Kisi("Mehmet", "SEZER", 45); </script>

4-) Ayrıca ECMAScript 5 ile birlikte gelen Object.create fonksiyonu ile nesne oluşturabiliriz.

const arabaPrototipi = {

marka: "Toyota",

model: "Corolla",

yil: 2023,

bilgileriYazdir() {

console.log(`Marka: ${this.marka}`);

console.log(`Model: ${this.model}`);

console.log(`Yıl: ${this.yil}`);

}

};

const araba1 = Object.create(arabaPrototipi);

araba1.renk = "Beyaz";

const araba2 = Object.create(arabaPrototipi);

araba2.renk = "Siyah";

araba1.bilgileriYazdir();

// Marka: Toyota

// Model: Corolla

// Yıl: 2023

araba2.bilgileriYazdir();

// Marka: Toyota

// Model: Corolla

// Yıl: 2023

console.log(araba1.marka); // "Toyota" yazdırır

console.log(araba2.renk); // "Siyah" yazdırır

**JavaScript this anahtar kelimesi**

JavaScript nesne kurucusu içindeki this anahtar kelimesi nesne özellik ve fonksiyonlarına erişim için kullanılır.

Nesne kurucu fonksiyonunun dönüş değeri this anahtar kelimesi kullanılırsa dönüş değeri nesnenin kendisi olur.

JavaScript this değişken değildir.

Anahtar kelimedir.

Değeri değiştirilemez.

JavaScript kurucuları constructor

<script>

var x1 = new Object(); // Yeni Object nesnesi

var x2 = new String(); // Yeni String nesnesi

var x3 = new Number(); // Yeni Number nesnesi

var x4 = new Boolean(); // Yeni Boolean nesnesi

var x5 = new Array(); // Yeni Array nesnesi

var x6 = new RegExp(); // Yeni RegExp nesnesi

var x7 = new Function(); // Yeni Function nesnesi

var x8 = new Date(); // Yeni Date nesnesi

</script>

JavaScript Math nesnesinin özellik ve fonksiyonları global olduğundan new anahtar kelimesi ile oluşturulmaz.

**JavaScript Private değişken tanımlama**

Alt çizgi notasyonu

JavaScript’te private değişken için özel bir tanım yoktur.

JavaScript geliştiricileri ortak olarak alt çizgi ile başlayan değişkenleri private değişken olarak kabul eder.

<script> "use strict";

function Nokta(x, y) {

this.\_x = x; // \_x private değişken

this.\_y = y; // \_y private değişken

this.getX = function() { return this.\_x; }; // dolaylı erişim

this.setX = function(x) { this.\_x = x; }; // dolaylı değer değiştirme

this.getY = function() { return this.\_y; }; // dolaylı erişim

this.setY = function(y) { this.\_y = y; }; // dolaylı değer değiştirme

}

var xNokta = new Nokta(0, 0);

xNokta.setX(10);

console.log(xNokta.getX());

xNokta.setY(20);

console.log(xNokta.getY());

console.log(xNokta.\_x); // 10

console.log(xNokta.\_y); // 20 </script>

Ancak bu yöntem pek güvenilir değildir.

Tam olarak Private değişken tanımına uymamaktadır.

Dışarıdan müdahale edilebilir.

**Object.defineProperty**

Object.defineProperty bir nesneye yeni bir özellik eklemek ve özelliğin okunabilir, silinebilir ve yazılabilir olma gibi kuralları belirmek için kullanılır.

JavaScript tanımlı tüm fonksiyon ve değişkenler window nesnesine kaydedilir.

Global bir sabit oluşturmak için window nesnesine yeni bir özellik ekleyerek sabit değerli bir değişken oluşturabiliriz.

JavaScript Object.defineProperty ile tanımlanmış değişkenlere ayrıca get ve set metotlarını tanımlayabiliriz.

JavaScript Object.defineProperty ile tanımlanmış bir değişkene değer atadığımızda set metodunu, değişkenin değerini okumaya çalıştığımızda get metodunu çağırır.

**Object.freeze**

Paramete olarak aldığı nesneye ait özelliklerin değiştirilmesini engeller.

**JavaScript Extension Metotlar**

JavaScript’e ait nesnelere yeni metot eklemek isteyebiliriz.

Bu ekleme işlemine genişletme / extension denir.

Tüm nesnelerin kendine ait bir prototype özelliği vardır.

**JavaScript Chaining Metotlar**

Chaining metodun amacı nesneye ait fonksiyonları sürekli nesneAdi.metotAdi gibi çağırmak yerine nesneAdi.metotAdi().metotAdi() gibi bir kullanımla çağırma imkanı verir.

Her bir metodun zincir halkası gibi birbirine bağlı olmasından dolayı bu yönteme Zincirleme metot denir.

**JavaScript Kalıtım Almak**

İki ayrı nesnenin özellik ve metotlarını bir nesneye toplama işlemine kalıtım almak veya miras alma olarak geçmektedir.

JavaScript kalıtım alma diğer kalıtım alma desteği sunan yazılım dillerinden farklıdır.

JavaScript ile kalıtım almak için kullanılan çeşitli yöntemler vardır.

**Constructor stealing – Kurucu çağırma**

Bu yöntemde kalıtım alacak nesne, kalıtım alınacak nesnenin kurucusunu call veya apply metodunu kullanarak çağırır.

<script> "use strict";

function Hayvan() {

this.renk = ["siyah", "beyaz"];

}

function Kedi() {

Hayvan.call(this); // Constructor stealing - Yapıcı çalma

//Hayvan.apply(this); // Constructor stealing </script>

**Prototype Özelliği:**

JavaScript'te, her nesnenin prototype adı verilen bir özelliği vardır. Prototype, bir nesnenin miras aldığı özelliklerin ve fonksiyonların bir kalıbını içerir. Bir nesne bir prototypeden türediğinde, prototipteki tüm özellikler ve fonksiyonlar türemiş nesne tarafından da kullanılabilir.

!!! constructor’lar object’leri ilk kullanıma hazırlamaktadır. İlk değeri atamak için.

Kalıtım alan: Kalıtım alınan

İnstanceof:

instanceof operatorü JavaScript'te bir nesnenin belirli bir constructor ile oluşturulup oluşturulmadığını kontrol etmek için kullanılır.

Nasıl Çalışır:

instanceof operatorünün sol tarafına bir nesne, sağ tarafına ise bir constructor yazılır.

instanceof operatorü, nesnenin prototype zincirini kontrol eder.

Prototype zincir, bir nesnenin miras aldığı özelliklerin ve fonksiyonların bir yoludur.

Eğer nesnenin prototype zinciri içerisinde sağ taraftaki constructor bulunursa, instanceof operatorü true değerini döndürür. Aksi takdirde false değeri döndürür.

console.log(kedi instanceof Hayvan); // true yazdırır

console.log(kopek instanceof Hayvan); // true yazdırır

Inheritance var olan class’ları kullanarak, yeni class’lar tanımlanmasına olanak sağlamaktadır.

Javascript’te Inheritance, prototype tabanlıdır. Yani aslında Javascript’te önceden tanımlı yada kullanıcı tarafından tanımlanan tüm object’ler, object nesnesinin prototype özelliği ile erişilebilen prototype nesnesinden türetilmektedir. Javascript’te bir fonksiyon oluşturulduğunda prototype özelliği ilgili fonksiyona otomatik olarak eklenir. Fonksiyonlarında birer object olduklarını unutmamak lazım.

PROTOTYPE

Rahatça eklemeler yapabilip hem de her şeyi değiştirmek zorunda kalamasak iyi olmaz mıydı ?

Her fonksiyon oluşturduğumuzda JavaScript motoru buna ekstra olarak bir property ekler. Eklenen property prototype dır. Fonksiyonun prototype objesine fonksiyonAdı.prototype şeklinde ulaşabiliriz.

**Programlama Dillerinde Ön Tanımlı Nesneler**

Programlama dillerinde, daha kolay ve hızlı bir şekilde kod yazmak için önceden tanımlanmış birçok nesne bulunur.

Matematiksel işlemler için: Math nesnesi

Tarih ve saat ile ilgili işlemler için: Date nesnesi

Dizi oluşturmak ve yönetmek için: Array nesnesi

Nesne oluşturmak ve yönetmek için: Object nesnesi

Metin işlemleri için: String nesnesi

Düzenli ifadeler ile metin arama ve değiştirme için: RegExp nesnesi

Fonksiyon oluşturmak ve yönetmek için: Function nesnesi

Hatalar ile ilgili işlemler için: Error nesnesi

**Kütüphane, Framework, Object Class ve Ön Tanımlı Nesneler**

Kütüphane:

Bir programlama dilinde yaygın olarak kullanılan fonksiyonlar ve değişkenler içeren bir kod koleksiyonudur.

Programcıların belirli görevleri daha kolay ve hızlı bir şekilde yerine getirmesine yardımcı olur.

Matematiksel işlemler, veri girişi/çıkışı, grafik oluşturma gibi birçok farklı alanda kütüphaneler bulunur.

Örnek:

JavaScript'te Math kütüphanesi matematiksel işlemler için kullanılır.

Python'da os kütüphanesi işletim sistemi ile etkileşim kurmak için kullanılır.

Framework:

Bir programın temel yapısını ve işleyişini tanımlayan bir yazılım altyapısıdır.

Programcıların belirli bir mimariye göre kod yazmalarını ve uygulamalar geliştirmelerini sağlar.

Web uygulamaları, mobil uygulamalar, masaüstü uygulamaları gibi farklı alanlarda framework'ler bulunur.

Örnek:

JavaScript'te React framework'ü web uygulamaları geliştirmek için kullanılır.

Java'da Spring Boot framework'ü web servisleri geliştirmek için kullanılır.

Object Class:

Bir programlama dilinde nesne oluşturmak için kullanılan bir kalıptır.

Nesnenin özelliklerini ve davranışlarını tanımlar.

Programcılar, Object Class'ı miras alarak kendi nesnelerini oluşturabilirler.

Örnek:

Java'da Object sınıfı tüm nesnelerin temel sınıfıdır.

Python'da object sınıfı tüm nesnelerin temel sınıfıdır.

Ön Tanımlı Nesneler:

Bir programlama dilinde önceden tanımlanmış ve kullanıma hazır nesnelerdir.

Programcıların yaygın görevleri daha kolay ve hızlı bir şekilde yerine getirmesine yardımcı olur.

Math nesnesi, Date nesnesi, Array nesnesi gibi birçok farklı ön tanımlı nesne bulunur.

Farklılıklar:

Kavram Tanım Örnek

Kütüphane Fonksiyonlar ve değişkenler içeren kod koleksiyonu Math kütüphanesi

Framework Bir programın temel yapısını tanımlayan yazılım altyapısı React framework'ü

Object Class Nesne oluşturmak için kullanılan kalıp Object sınıfı

Ön Tanımlı Nesneler Önceden tanımlanmış ve kullanıma hazır nesneler Math nesnesi

Özet:

Kütüphaneler ve framework'ler, programcıların kod yazma ve uygulama geliştirme sürecini kolaylaştırır.

Object Class'lar, nesne tabanlı programlamada temel bir rol oynar.

Ön tanımlı nesneler, programcıların yaygın görevleri daha hızlı ve kolay bir şekilde yerine getirmesine yardımcı olur.

**Framework'ler Nasıl Çalışır?**

Framework'ler, belirli bir programlama dili veya teknolojisi üzerinde inşa edilmiş ve geliştiricilere temel yapı taşları sunan yazılım altyapılarıdır. Temel işleyişleri şu şekilde özetlenebilir:

1. Soyutlama:

Framework'ler, karmaşık işlemleri ve kavramları soyutlayarak geliştiricilerin daha basit ve anlaşılır bir şekilde kod yazmalarını sağlar. Örneğin, bir web uygulaması framework'ü, HTTP protokolü ile uğraşmak yerine, geliştiricilerin HTML ve JavaScript gibi web teknolojilerine odaklanmasına imkan verir.

2. Kod Tekrarını Önleme:

Framework'ler, sık kullanılan kodları ve işlevleri önceden hazırlayarak geliştiricilerin tekerleği yeniden icat etmesini önler. Bu sayede kod tekrarı azalır, geliştirme süreci kısalır ve kod daha tutarlı hale gelir.

3. Yapı ve Organizasyon:

Framework'ler, uygulamaların belirli bir mimariye göre geliştirilmesini sağlayarak kodun daha organize ve bakımı kolay olmasını sağlar. Örneğin, bir MVC framework'ü, uygulamayı model, görünüm ve denetleyici katmanlarına böler ve her katmanın sorumluluklarını tanımlar.

4. Hızlı Geliştirme:

Framework'ler, önceden hazırlanmış bileşenler ve şablonlar sunarak geliştiricilerin prototip oluşturma ve uygulamaları daha hızlı bir şekilde geliştirmelerine yardımcı olur.

5. Eklentiler ve Kütüphaneler:

Çoğu framework, geliştiricilerin ihtiyaç duydukları ek işlevleri eklemelerine imkan veren geniş bir eklenti ve kütüphane yelpazesine sahiptir.

**Single Page Application(SPA):**

Tek sayfalık uygulamalar, kullanıcı arayüzünün tek bir sayfadan oluştuğu web uygulamalarıdır. Bu sayfada dinamik içerikler gösterilir ve kullanıcı sayfalar arasında geçiş yapmadan uygulama ile etkileşime girer.

SPA'larda Kullanılan Teknolojiler:

JavaScript: SPA'ların temelini oluşturur.

AJAX: Sayfa yenilemeden dinamik içerik yüklemek için kullanılır.

Routing: Kullanıcıyı farklı uygulama bölümlerine yönlendirmek için kullanılır.

State Management: Uygulamanın durumunu yönetmek için kullanılır.

Popüler SPA Framework'leri:

React: Popüler bir SPA framework'üdür.

Angular: Bir diğer popüler SPA framework'üdür.

Vue.js: Daha hafif bir SPA framework'üdür.

SPA'ya Örnekler:

Gmail

Facebook

Twitter

Spotify

**State Management:** Tek sayfalık uygulamalarda, durumu yönetmek için kullanılır. Kullanıcı etkileşimleri sonucunda durum sürekli değişir.

Tek sayfalık uygulamalarda (SPA) kullanıcı arayüzünün durumu (state) sürekli değişir. Bu durum, kullanıcının etkileşimleri, ağ istekleri veya diğer olaylar sonucunda güncellenebilir. State management, bu durumu yönetmek ve tüm bileşenlerin durum değişikliklerinden haberdar olmasını sağlamak için kullanılan bir tekniktir.

**Routing:**

Routing, tek sayfalık uygulamalarda (SPA) kullanıcıyı farklı uygulama bölümlerine yönlendirmek için kullanılan bir tekniktir. Kullanıcı, URL'yi değiştirerek veya uygulama içerisindeki bağlantıları tıklayarak farklı sayfalara geçiş yapabilir.

**Abstract Class**

Abstract class, ortak özellikleri olan nesneleri bir çatı altında toplamak için kullanılır.

Sınıfın önüne “abstract” sözcüğü yazılarak soyutlaştırma işlemi yapılır. Abstract sınıftan kalıtım almak için de “extends” kullanılır.

Genellikle Base Class tanımlamak için kullanılır.

**Abstract ve Interface Kavramları**

Abstract:

Soyut bir sınıftır.

Nesne oluşturmak için kullanılamaz.

Miras alma (inheritance) için kullanılır.

Soyut (abstract) metotlar ve somut (concrete) metotlar içerebilir.

Soyut metotlar gövdesizdir, nasıl işleyeceğini tanımlamaz.

Somut metotlar gövdelidir, nasıl işleyeceğini tanımlar.

Interface:

Soyut bir sınıfa benzer.

Nesne oluşturmak için kullanılamaz.

Miras alma (inheritance) için kullanılır.

Yalnızca soyut metotlar içerir.

Soyut metotların gövdeleri tanımlanmaz.

public abstract class Hayvan {

public abstract void sesCikar();

public void beslenir() {

System.out.println("Hayvan besleniyor.");

}

}

public class Kedi extends Hayvan {

@Override

public void sesCikar() {

System.out.println("Miyav!");

}

}

public class Kopek extends Hayvan {

@Override

public void sesCikar() {

System.out.println("Hav!");

}

}

**Abstract class ve kalıtım**

Abstract class:

Ortak özellikleri ve davranışları tanımlayan bir sınıftır.

Nesne oluşturmak için kullanılamaz.

Miras alma (inheritance) için kullanılır.

Soyut (abstract) ve somut (concrete) metotlar içerebilir.

Kalıtım:

Bir sınıftan diğerine özelliklerin ve davranışların aktarılmasıdır.

Kod tekrarını önler ve kodun daha organize olmasını sağlar.

Alt sınıflar (derived classes), üst sınıflardan (base classes) miras alır.

Abstract class ve kalıtım arasındaki farklar:

Abstract class bir sınıftır, kalıtım bir mekanizmadır.

Abstract class nesne oluşturmak için kullanılamazken, kalıtım yoluyla nesneler oluşturulabilir.

Abstract class ortak özellikleri ve davranışları tanımlar, kalıtım bu özellikleri ve davranışları aktarır.

Örnek:

Java

public abstract class Hayvan {

public abstract void sesCikar();

public void beslenir() {

System.out.println("Hayvan besleniyor.");

}

}

public class Kedi extends Hayvan {

@Override

public void sesCikar() {

System.out.println("Miyav!");

}

}

public class Kopek extends Hayvan {

@Override

public void sesCikar() {

System.out.println("Hav!");

}

}

**Abstract Interface Ne Zaman Kullanılır?**

1. Ortak Davranışı Tanımlamak: Birden fazla sınıfın ortak davranışını tanımlamak için kullanılır. Bu, kod tekrarını önler ve kodun daha organize olmasını sağlar.

2. Soyutlama Sağlamak: Sınıfların nasıl çalıştığını gizlemek ve kodun daha anlaşılır ve bakımı kolay olmasını sağlamak için kullanılır.

3. Polimorfizmi Uygulamak: Polimorfizmi uygulamak ve kodun daha esnek ve genişletilebilir olmasını sağlamak için kullanılır.

**Abstract ve Interface'in Hangi Durumda Kullanılacağına Karar Verme:**

Ortak özelliklerin ve davranışların somut bir şekilde nasıl işleneceği belliyse abstract class kullanılır.

Ortak özelliklerin ve davranışların somut bir şekilde nasıl işleneceği belli değilse interface kullanılır.

Bir sınıftan birden fazla miras almanız gerekiyorsa interface kullanılır.

Daha fazla esnekliğe ihtiyacınız varsa interface kullanılır.

**Soyutlama ve Polimorfizm Kavramları**

Soyutlama:

Gerçek dünyadaki karmaşık nesneleri ve işlemleri, daha basit ve anlaşılır bir şekilde modelleme yöntemidir.

Önemsiz ayrıntıları göz ardı ederek, asıl önemli olan unsurlara odaklanmayı sağlar.

Kodun daha anlaşılır ve bakımı kolay olmasını sağlar.

Polimorfizm:

Aynı kodun farklı durumlarda farklı şekilde davranmasını sağlayan bir mekanizmadır.

Kalıtım (inheritance) ve overloading (aşırı yükleme) ile sağlanır.

**static:**

Bir değişken veya metodun sınıfın tüm örnekleri tarafından paylaşılmasını sağlar.

Genellikle sınıfa ait sabit değerler (örneğin matematiksel sabitler) için kullanılır.

static değişkenlerin değerleri değiştirilebilir.

public class Araba {

public static final int MAX\_HIZ = 200; // static final değişken

public static void calistir() {

System.out.println("Araba çalıştı.");

}

}

Matematiksel sabitler (PI, e gibi)

Sınıf sayacı (kaç tane nesne oluşturulduğunu tutmak için)

Yardımcı fonksiyonlar (sınıfa ait herkesin kullanabileceği fonksiyonlar)

**final:**

Bir değişkenin değerinin atanması sonrasında değiştirilemez olmasını sağlar.

Değişkene sadece bir kere değer atanabilir.

Bu terim genellikle sabit değerler tanımlamak için kullanılır.

Sabit değerler (değişmeyecek değerler)

Nesne oluşturulduktan sonra değiştirilmemesi gereken değerler

**static ve final birlikte kullanıldığında:**

Değişken sınıfın tüm örnekleri tarafından paylaşılır ve değiştirilemez.

Genellikle matematiksel sabitler gibi değişmeyen sabit değerler tanımlamak için kullanılır.

**const ve final Arasındaki Farklar**

Değişken Değeri:

const: Değişken derleme zamanında sabit bir değerle başlatılmalıdır.

final: Değişken çalışma zamanında bir değerle başlatılabilir.

Erişim:

const: Her yerde kullanılabilir.

final: Sadece tanımlandığı sınıfta ve alt sınıflarda kullanılabilir.

**Field: private String degisken;**

Bir field, bir sınıfın içinde doğrudan tanımlanmış basit bir veri üyesidir.

Nesneyle ilişkili veriyi doğrudan depolar.

Erişime izin verilen seviyeye bağlı olarak private, public, protected veya package-private olabilir.

Varsayılan olarak privatedır, yani sadece sınıfın içinden erişilebilirler.

**Property: private String degisken; + getter, setter metotları**

Bir property, bir field'a erişimi ve potansiyel olarak onu değiştirmeyi kontrol eden daha üst düzey bir yapıdır.

Genellikle bir getter ve opsiyonel olarak bir setter metodundan oluşur.

Getter metodu, ilişkili field'ın değerini okumanıza izin verir.

Setter metodu (varsa), ilişkili field'ın değerini değiştirmenize izin verir, genellikle ek mantık veya doğrulama ile birlikte kullanılır.

!!! Constructor, bir nesnenin ilk kez oluşturulduğunda çalışan özel bir metottur.

**Nesneye Dayalı Programlamada İlişkiler: "Is-a", "Has-a", "Uses-a" ve "Composition"**

**1. "Is-a" (Tür-Alt Tür) İlişkisi:**

Kalıtım yoluyla kurulan bir ilişkidir.

Alt sınıf (child class), üst sınıftan (parent class) miras yoluyla özellik ve davranışlarını alır.

Alt sınıf, üst sınıfın bir "türevi" olarak kabul edilir.

public class Hayvan {

public void yemek() {

System.out.println("Yiyor...");

}

}

public class Kedi extends Hayvan {

public void miyavlamak() {

System.out.println("Miyavlıyor...");

}

}

Bu örnekte:

Kedi sınıfı, Hayvan sınıfından türemiştir.

Her Kedi, aynı zamanda bir Hayvan'dır.

Kedi sınıfı, Hayvan sınıfının yemek() metodunu miras alır ve kendi metodu miyavlamak()'ı ekler.

**2. "Has-a" (İçerme) İlişkisi:**

Bir sınıfın, başka bir sınıfın bir veya birden fazla nesnesini içermesi durumunda kullanılır.

Bu ilişki, nesnelerin bir bütün olarak çalışmasını sağlar.

public class Araba {

private Motor motor; // Araba bir Motor nesnesi içerir.

private List<Lastik> lastikler; // Araba birden fazla Lastik nesnesi içerir.

}

Bu örnekte:

Araba sınıfı, bir Motor ve birden fazla Lastik nesnesi içerir.

**3. "Uses-a" (Kullanım) İlişkisi:**

Bir sınıfın, başka bir sınıfın hizmetlerinden faydalanması durumunda kullanılır.

Bu ilişki, kod tekrarını azaltır ve esneklik sağlar.

public class Araba {

private Motor motor; // Araba bir Motor nesnesi içerebilir (isteğe bağlı).

public void calistir() {

if (motor != null) {

motor.calistir(); // Araba, Motor sınıfının calistir() metodunu kullanır.

}

}

}

Bu örnekte:

Araba sınıfı, Motor sınıfının calistir() metodunu kullanarak motoru çalıştırır. Motor nesnesi opsiyoneldir.

**4. "Composition" (Oluşturma) İlişkisi:**

Bir sınıfın, başka bir sınıfın bir veya birden fazla nesnesini içermesi ve bu nesnelerin, sahibi olan sınıfın yaşam döngüsüne bağlı olması durumunda kullanılır.

Sahibi olan sınıf yok edilirse, içerdiği nesneler de yok edilir.

Daha güçlü bir "Has-a" ilişkisidir.

public class Motor {

private Silindir silindir; // Motor, bir Silindir nesnesi içerir.

public Motor(Silindir silindir) {

this.silindir = silindir; }

}

Bu örnekte:

Motor sınıfı bir Silindir nesnesi içerir ve Motor nesnesi oluşturulurken Silindir nesnesi de zorunlu olarak atanır.

Motor yok edilirse, Silindir nesnesi de yok edilir.

Composition ilişkisinde ise, içerilen nesnenin yaşam döngüsü, sahibi olan sınıfın yaşam döngüsüne bağlıdır. Bu, şu demektir:

1. Oluşturma:

Sahibi olan sınıfın constructor'ı çalıştırıldığında, içerilen nesne de otomatik olarak oluşturulur.

2. Yok Edilme:

Sahibi olan sınıf yok edildiğinde, içerilen nesne de otomatik olarak yok edilir.

3. Bağımsızlık:

İçerilen nesne, sahibi olan sınıftan bağımsız olarak var olamaz.

Örnek:

Java

public class Motor {

private Silindir silindir; // Composition ilişkisi

public Motor() {

silindir = new Silindir(); // Silindir nesnesi constructor'da oluşturulur.

}

@Override

protected void finalize() throws Throwable {

super.finalize();

silindir = null; // Sahibi olan sınıf yok edilirken, Silindir nesnesi de yok edilir.

}

}

Bu örnekte:

Motor sınıfı bir Silindir nesnesi içerir.

Motor nesnesi oluşturulduğunda, Silindir nesnesi de otomatik olarak constructor'da oluşturulur.

Motor nesnesi yok edildiğinde, Silindir nesnesi de otomatik olarak finalize() metodunda yok edilir.

**finalize() Metodunun İşlevi**

finalize() metodu, nesneye dayalı programlamada (OOP) bir nesnenin yok edilmeden önce çalıştırılan özel bir metottur.

try-with-resources: Kaynakların otomatik olarak serbest bırakılması için try-with-resources blokları kullanılabilir.

Cleaner API: Java 9'dan itibaren, finalize() metoduna daha güvenli bir alternatif sunan Cleaner API kullanılabilir.

**ES6 let**

JavaScript EcmaScript 6 ile birlikte değişken oluşturma ve tanımlama için kullanılan let, kullanım alanı tam olarak anlaşılmayan veya tam olarak ne olduğu açıklanmayan değişken oluşturma ve tanımlama anahtar kelimesi diyebiliriz.

**JavaScript Değişken Kaldıraç Özelliği**

JavaScript'te değişken kaldıraç özelliği, bir değişkenin kapsamını (scope) belirleme yeteneğidir. Bu özellik, var, let ve const anahtar kelimeleri ile kontrol edilir.

Kaldıraç Seviyeleri:

Blok Kaldıraç: Bir değişken, if bloğu, for döngüsü veya function gibi bir blok içinde tanımlanırsa, sadece o blokta erişilebilir.

İşlev Kaldıraç: Bir değişken, bir fonksiyon içinde tanımlanırsa, sadece o fonksiyon ve iç içe geçmiş fonksiyonlar tarafından erişilebilir.

Global Kaldıraç: Bir değişken, herhangi bir bloğun veya fonksiyonun dışında tanımlanırsa, tüm kod tarafından erişilebilir.

Anahtar Kelimeler:

var: Değişkenin yeniden atanmasına izin verir.

let: Değişkenin sadece bir kere atanmasına izin verir ve blok kapsamındadır.

const: Değişkenin sadece bir kere atanmasına izin verir ve sabittir, değiştirilemez.

**ES6 Default Parameters**

<script> "use strict";

Date.prototype.getYas = function(dogumYili) {

return this.getFullYear() - dogumYili;

}

var tarih = new Date();

console.log(tarih.getYas(1970)); </script>

Fonksiyona parametre girilmediği zaman fonksiyon çalışmaz hale gelecek veya yanlış bir döndürecektir.

Fonksiyona varsayılan parametre değeri yazalım.

<script> "use strict";

Date.prototype.getYas = function(dogumYili = 1950) {

return this.getFullYear() - dogumYili;

}

var tarih = new Date();

console.log(tarih.getYas(1970)); // 1970 gönderildi </script>

**ES6 Destructuring Assignment**

JavaScript dizi içerisindeki değerleri ayrı ayrı değişkenlere atamak oldukça zahmetli ve zaman alanbir durumdu.

<script> "use strict";

var sayilar = [1, 2, 3, 4, 5];

var [a,,b,,c] = sayilar;

console.log(a,b,c); // 1 3 5 </script>

!!!!Sabitler büyük harf ile isimlendirilirler. Eğer birden fazla kelimeden oluşuyorsa “\_” ile bu kelimeleri ayırmak mümkündür.

const RENK\_KIRMIZI = "#F00";

**Tip Dönüşümleri: 3 tane vardır. Boolean ,toString, Number**

toString

Number

Boolean

**Ok Fonksiyonları**

let topla = (a, b) => a + b;

ok fonksiyonu aşağıdaki fonksiyon ifadesinin daha özlü yazılmış halidir.

let topla = function(a, b) {

return a + b;

};

**Kesme noktası**, JavaScript çalışırken çalışmasını o noktada durdurmasını sağlar.

Kod durdurulduğunda, o anki değişken değerlerini inceleyebilir veya konsoldan kod çalıştırabilirsiniz. Diğer bir deyişle hata ayıklayabilirsiniz.

debugger kodu ile de hata ayıklama işlemini yapmak mümkündür.

console.log fonksiyonunu kullanabilirsiniz.

**Özellik Bayrakları**

Obje özellikleri değer dışında, 3 özelliğe sahiptir ( bunlara “bayraklar” denir. )

yazılabilir – eğer true ise değiştirilebilir aksi halde sadece okunabilir.

sayılabilir – eğer true ise döngü içinde listelenmiştir, aksi halde listelenmemiştir.

ayarlanabilir – eğer true ise özellik silinebilir ve nitelikler ( attributes ) değiştirilebilir, diğer türlü değiştirilemez.

**Object.getOwnPropertyDescriptor** metodu bir özellik hakkındaki tüm bilgilerin sorgulanabilmesini sağlar.

Yazımı:

let descriptor = Object.getOwnPropertyDescriptor(obj, propertyName);

obj

Bilgi alınacak obje

propertyName

Özelliğin ismi

let user = {

name: "John"

};

let descriptor = Object.getOwnPropertyDescriptor(user, 'name');

alert( JSON.stringify(descriptor, null, 2 ) );

/\* property descriptor:

{

"value": "John",

"writable": true,

"enumerable": true,

"configurable": true

} \*/

Bayrakları değiştirmek için Object.defineProperty kullanılabilir.

Yazımı:

**Object.defineProperty**(obj, propertyName, descriptor)

obj, propertyName

Üzerinde çalışılacak obje ve özellik.

descriptor

Uygulanacak özellik tanımlayıcı

let user = {};

Object.defineProperty(user, "name", {

value: "John"

});

let descriptor = Object.getOwnPropertyDescriptor(user, 'name');

alert( JSON.stringify(descriptor, null, 2 ) );

/\*{

"value": "John",

"writable": false,

"enumerable": false,

"configurable": false

} \*/

**Protitipsel Kalıtım**

Programlarken genelde bir şeyi alır ve bunu genişletmek isteriz.

Örneğin, kullanici adında bir obje ve bunun özellikleri ve metodları olsun, bunu biraz düzenleyerek admin ve misafir gibi iki farklı obje oluşturmak isteriz. Yani kullanici objesini doğrudan kopyalamak veya metodlarını tekrardan uygulamak değil bunlar üzerinden yeni objeler yaratmak isteyebiliriz.

Prototip kalıtımı buna olanak sağlamaktadır.

[[Prototype]]

Javascript objeleri gizli bir özellik olan [[Prototype]] özelliğine sahiptirler. Bu null olabilir veya başka objeye referans verebilir. Referans verilen obje “prototip” olarak adlandırılır.

[Prototip]]'in “büyülü” bir anlamı bulunmaktadır. Objeden bir özellik okunmak istendiğinde, ve bu obje bulunamadığında JavaScript bunu otomatik olarak prototip’ten alır. Programlamada buna prototip kalıtımı denir. Birçok dil özelliği ve programlama tekniği bunun üzerine kuruludur.

[[Prototpe]] gizli bir özelliktir, fakat bunu ayarlamanın birçok yolu vardır.

Bunlardan biri \_\_proto\_\_ kullanmaktır:

let animal = {

eats: true

};

let rabbit = {

jumps: true

};

rabbit.\_\_proto\_\_ = animal;

**Fonksiyon Türleri: 4 tanedir.**

**İsimlendirmelerine Göre Fonksiyon Türleri:** (Regular/Named Functions, Anonymous Functions)

Normal Fonksiyonlar (Regular/Named Functions): İsimleri olan ve kodda açıkça tanımlanan fonksiyonlardır.

Anonim Fonksiyonlar (Anonymous Functions): İsimleri olmayan ve genellikle değişkenlere atanan veya doğrudan fonksiyon çağrılarında kullanılan fonksiyonlardır.

**İşlevlerine Göre Fonksiyon Türleri:** (Pure Functions, Constructor Functions, IIFE, High Order Functions)

Saf Fonksiyonlar (Pure Functions): Giriş parametreleri aynıysa her zaman aynı sonucu üreten fonksiyonlardır. Yan etkileri yoktur.

Yapıcı Fonksiyonlar (Constructor Functions): Yeni nesneler oluşturmak için kullanılan fonksiyonlardır.

Hemen Çalıştırılan Fonksiyonlar (IIFE - Immediately Invoked Function Expressions): Tanımlandıkları anda otomatik olarak çalıştırılan fonksiyonlardır.

Yüksek Düzey Fonksiyonlar (High Order Functions): Fonksiyon parametresi olarak alan veya fonksiyon döndüren fonksiyonlardır.

**İşleyiş Türlerine Göre:** (Asynchronous Functions, Synchronous Functions, Generator Functions)

Asenkron Fonksiyonlar (Asynchronous Functions): İşlemlerini tamamlamak için bir süre beklemesi ge Senkron Fonksiyonlar (Synchronous Functions): İşlemlerini tamamlamak için herhangi bir bekleme süresine ihtiyaç duymayan ve programın akışını engelleyen fonksiyonlardır.reken ve bu süre boyunca programın diğer kısımlarını engellemeyen fonksiyonlardır.

Jeneratör Fonksiyonlar (Generator Functions): Birden fazla değer üretebilen ve her seferinde bir sonraki değeri döndüren fonksiyonlardır.

**Gösterimlerine Göre Fonksiyon Türleri:** (Normal Functions, Arrow Functions)

Normal Fonksiyonlar (Normal Functions): Anahtar kelime ve fonksiyon ismini içeren geleneksel fonksiyon tanımlama yöntemidir.

Ok Fonksiyonları (Arrow Functions): Daha kısa ve öz bir fonksiyon tanımlama yöntemi sunar. Ok (=>) sembolünü kullanır.

**Modül nedir?**

Modül sadece bir dosyadır, tek bir program. export ve import yönergeleri modüller arasında işlevselliğin yer değiştirmesini sağlar.

**Asenkron Metotlar**

Asenkron metotlar, bir işlemi tamamlamak için bekleme yapmadan programın akışının devam etmesine izin veren özel fonksiyonlardır. Bu metotlar, işlemi arka planda yürütür ve tamamlandığında bir geri bildirim gönderirler.

Bir işlem başlatırsınız ve program akışı o işlemi beklerken durmak yerine diğer işleri yapmaya devam eder. İşlem tamamlandığında size bir haber verilir ve siz de işlemin sonucuna göre programınızı yönlendirirsiniz.

Asenkron Metotlarda Kullanılan Kavramlar

*Callback:*

İşlem tamamlandıktan sonra tetiklenen fonksiyondur.

Asenkron metotlar, işlemin sonucunu ve durumunu callback fonksiyonuna parametre olarak gönderir.

Callback fonksiyonu, bu parametreleri kullanarak işlemin sonucuna göre program akışını yönlendirir.

veriAl fonksiyonu, veri alma işlemini başlatır ve tamamlandığında callback fonksiyonunu tetikler.

callback fonksiyonu, err ve data parametrelerini alır.

err parametresi, hata durumunda hata mesajını içerir.

data parametresi, başarı durumunda alınan veriyi içerir.

*Promise:*

Bir işlemin tamamlanma durumunu ve sonucunu temsil eden bir nesnedir.

Promise nesneleri, then ve catch metotları ile işlenir.

then metodu, işlemin başarıyla tamamlanması durumunda tetiklenir.

catch metodu, işlemin hata ile sonuçlanması durumunda tetiklenir.

JavaScript

const veriAlPromise = fetch("https://api.example.com");

veriAlPromise

.then(response => response.json())

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

.catch(err => console.error(err));

Açıklama:

veriAlPromise değişkeni, fetch fonksiyonu ile bir Promise nesnesi oluşturur.

then metodu, Promise'in başarıyla tamamlanması durumunda tetiklenir ve JSON formatında gelen veriyi işler.

catch metodu, Promise'in hata ile sonuçlanması durumunda tetiklenir ve hata mesajını yazdırır.

*Async/Await:*

async anahtar kelimesi, bir fonksiyonun asenkron olduğunu belirtir.

await anahtar kelimesi, bir asenkron işlemin tamamlanmasını bekler.

async function veriAlAsync() {

const response = await fetch("https://api.example.com");

const data = await response.json();

console.log(data);

}

veriAlAsync();

veriAlAsync fonksiyonu, async anahtar kelimesi ile asenkron olarak tanımlanır.

await anahtar kelimesi, fetch ve json işlemlerinin tamamlanmasını bekler.

İşlemler tamamlandıktan sonra, alınan veri konsola yazdırılır.

**Proxy:**

Bir nesnenin önüne koyduğumuz ve onu kontrol etmemizi sağlayan bir "kapı görevlisi" gibidir.

Kapıdan kimin girip çıkacağını, neleri alıp götürebileceklerini kontrol edebiliriz.

Bu sayede nesnenin hangi özelliklerine erişilebileceğini, hangi değerlerin atanabileceğini ve nesnenin nasıl kullanılacağını belirleyebiliriz.

**Reflect:**

Proxy'nin "dil tercümanı" gibidir.

Proxy'nin karmaşık dilini daha basit ve anlaşılır hale getirir.

Proxy'nin sunduğu işlevleri daha kolay kullanmamızı sağlar.

Örnek:

Diyelim ki bir mağazanız var ve ürünlerinizi satmak istiyorsunuz.

Proxy: Mağazanızın kapısıdır.

Kapı görevlisi: Müşterilerin hangi ürünlere bakabileceklerini, neleri satın alabileceklerini ve ne kadar ödeme yapmaları gerektiğini kontrol eder.

Dil tercümanı: Müşterilerin farklı dilleri konuşsa bile kapı görevlisiyle iletişim kurmalarını sağlar.

Proxy ve Reflect Örnek Kod

Örnek 1: Erişim Kontrolü

JavaScript

const hedefNesne = {

isim: "Ahmet",

yas: 30,

maas: 10000,

};

const handler = {

get(target, prop, receiver) {

if (prop === "maas") {

console.log("Maaş bilgisi gizlidir!");

return "Gizli";

}

return Reflect.get(target, prop, receiver);

}

};

const proxy = new Proxy(hedefNesne, handler);

console.log(proxy.isim); // "Ahmet" yazdırır

console.log(proxy.yas); // 30 yazdırır

console.log(proxy.maas); // "Maaş bilgisi gizlidir!" yazdırır

Açıklama:

Bu örnekte, hedefNesne nesnesinin maas özelliğine erişim kısıtlanmıştır.

handler nesnesinin get fonksiyonu, maas özelliğine erişildiğinde "Maaş bilgisi gizlidir!" mesajını yazdırır ve "Gizli" değerini döndürür.

Proxy nesnesi aracılığıyla maas özelliğine erişildiğinde, get fonksiyonu tetiklenir ve mesaj ve değer yazdırılır.

Örnek 2: Veri Doğrulama

JavaScript

const handler = {

set(target, prop, value, receiver) {

if (prop === "yas" && typeof value !== "number") {

throw new TypeError("Yaş sayı olmalıdır!");

}

return Reflect.set(target, prop, value, receiver);

}

};

const proxy = new Proxy({}, handler);

proxy.isim = "Ayşe";

proxy.yas = 25;

// proxy.yas = "yirmi beş"; // TypeError: Yaş sayı olmalıdır! hatası fırlatır

console.log(proxy.isim); // "Ayşe" yazdırır

console.log(proxy.yas); // 25 yazdırır

Proxy ve Reflect Örnek Kod

Örnek 1: Erişim Kontrolü

JavaScript

const hedefNesne = {

isim: "Ahmet",

yas: 30,

maas: 10000,

};

const handler = {

get(target, prop, receiver) {

if (prop === "maas") {

console.log("Maaş bilgisi gizlidir!");

return "Gizli";

}

return Reflect.get(target, prop, receiver);

}

};

const proxy = new Proxy(hedefNesne, handler);

console.log(proxy.isim); // "Ahmet" yazdırır

console.log(proxy.yas); // 30 yazdırır

console.log(proxy.maas); // "Maaş bilgisi gizlidir!" yazdırır

Kodu dikkatli kullanın.

Açıklama:

Bu örnekte, hedefNesne nesnesinin maas özelliğine erişim kısıtlanmıştır.

handler nesnesinin get fonksiyonu, maas özelliğine erişildiğinde "Maaş bilgisi gizlidir!" mesajını yazdırır ve "Gizli" değerini döndürür.

Proxy nesnesi aracılığıyla maas özelliğine erişildiğinde, get fonksiyonu tetiklenir ve mesaj ve değer yazdırılır.

Örnek 2: Veri Doğrulama

JavaScript

const handler = {

set(target, prop, value, receiver) {

if (prop === "yas" && typeof value !== "number") {

throw new TypeError("Yaş sayı olmalıdır!");

}

return Reflect.set(target, prop, value, receiver);

}

};

const proxy = new Proxy({}, handler);

proxy.isim = "Ayşe";

proxy.yas = 25;

// proxy.yas = "yirmi beş"; // TypeError: Yaş sayı olmalıdır! hatası fırlatır

console.log(proxy.isim); // "Ayşe" yazdırır

console.log(proxy.yas); // 25 yazdırır

Açıklama:

Bu örnekte, proxy nesnesinin yas özelliğine atanan değerin sayı olması zorunlu kılınmıştır.

handler nesnesinin set fonksiyonu, yas özelliğine değer atanırken kontrol yapar ve sayı değilse hata fırlatır.

proxy nesnesi aracılığıyla yas özelliğine sayı olmayan bir değer atanmaya çalışılırsa hata fırlatılır.

Örnek 3: İşlem Yakalama

JavaScript

const handler = {

get(target, prop, receiver) {

console.log(`"${prop}" özelliğine erişildi!`);

return Reflect.get(target, prop, receiver);

},

set(target, prop, value, receiver) {

console.log(`"${prop}" özelliğine "${value}" değeri atandı!`);

return Reflect.set(target, prop, value, receiver);

}

};

const proxy = new Proxy({}, handler);

proxy.isim = "Fatma";

proxy.soyisim = "Yılmaz";

console.log(proxy.isim); // "isim" özelliğine erişildi! // "Fatma" yazdırır

console.log(proxy.soyisim); // "soyisim" özelliğine erişildi! // "Yılmaz" yazdırır

Açıklama:

Bu örnekte, proxy nesnesi üzerinde yapılan işlemler (okuma, yazma) izlenir.

handler nesnesinin get ve set fonksiyonları, her işlemde bir mesaj yazdırır.

Bu sayede, nesnenin nasıl kullanıldığını takip etmek mümkün hale gelir.

**Cache-validation:**

Cache Nedir? Verileri geçici depolama, daha hızlı erişim için

Cache, verileri daha hızlı erişim için geçici olarak depolamak için kullanılan bir mekanizmadır. Bu, bir web tarayıcısının web sayfalarını ve resimleri belleğe alması veya bir uygulamanın veritabanı sorgularını önbelleğe alması gibi olabilir.

Validation Nedir? Validation, önbelleğe alınmış verilerin orijinal kaynakla tutarlı kalmasını sağlama işlemidir. Başka bir deyişle, önbelleğe alınmış bilginin hala güncel olduğundan emin olmaktır.

**Diziler:** birbirine benzeyen ve sınırlı sayıda olan verilere denir.

Javascriptte aynı değişkeni birden fazla tanımlayabiliriz.

**Döngüler:** birbirine benzeyen işlemleri arka arkaya yapan işlemler

**Javascript Html Dom:** document object model, html-----html, head,body şeklinde ilerler.

HTML DOM, HTML öğelerinin nasıl alınacağına, değiştirileceğine, ekleneceğine veya silineceğine ilişkin bir standarttır.

GetElementById();

GetElementsByTagName();

İnnerHTML; değeri okumak için kullanılır.

GetElementsByClassName(); css deki class kullanarak onları almak

QuerySelectorAll();

GetElementsByName();

AddEventListener(“click”); bir nesneye tıklayınca

AddEventListener(“mouseover”); bir nesnenin üzerine gelince