



JavaScript



Javascript

Introduction

- › Javascript nedir?
- › Compiler & Interpreter
- › Kullanım amaçları
- › Sürümleri
- › Popülarite
- › İlk kodlar



Javascript nedir?



Yüksek seviyeli, nesne yönelimli, yorumlayıcı tabanlı ve dinamik bir programlama dilidir.

Yüksek Seviyeli

Hafıza yönetimi gibi karmaşık görevleri düşünmemize gerek yoktur.

Nesne Yönelimli

Nesne özelliklerinin (kalıtım, çok şekillilik v.b.) kullanılmasına imkan sağlamaktadır.

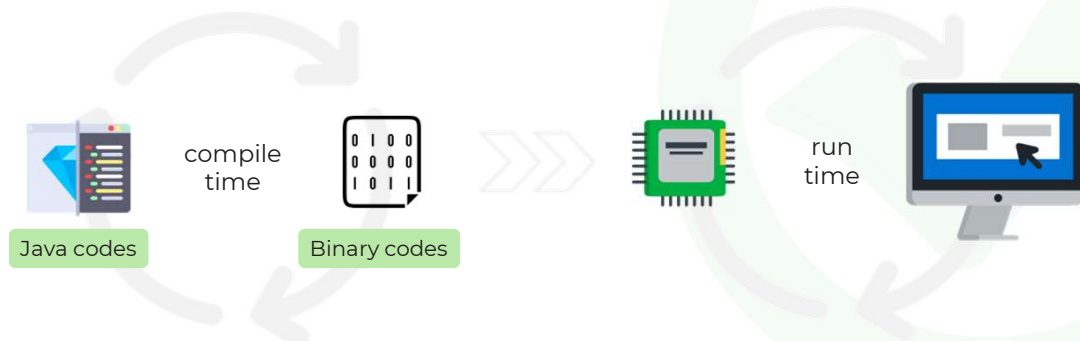
Yorumlayıcı Tabanlı

Derleyicide olduğu gibi tüm komutların bir kere de makine koduna çevirmek yerine tek-teke alınıp makine koduna çevrilip çalıştırılmasını sağlar.



Compiler vs Interpreter

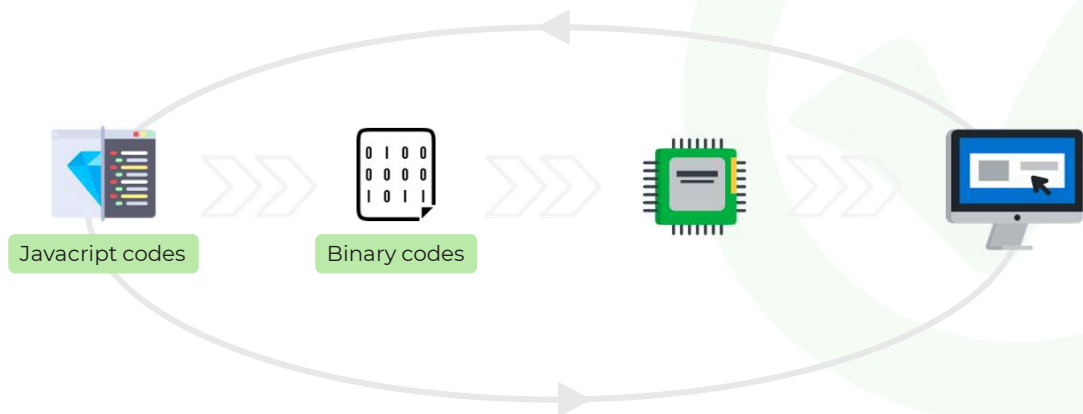
Compiler(Derleyici)





Compiler vs Interpreter

Interpreter (Yorumlayıcı)



Nerede kullanılır?

HTML



JAVASCRIPT



CSS





Neden Javascript

Front-end Dev



React, Svelte, Vue, Angular gibi JS framework ve kütüphaneleri ile web ve mobil uygulamalar geliştirilebilmektedir.

Facebook, Netflix, Dropbox, AirBnb

Back-end Dev



NodeJs ile sunucu tarafında uygulamalar geliştirilebilmektedir.

Netflix, Uber, E-Bay



Neden Javascript

Mobile Dev



React Native, Ionic, Native Script gibi kütüphaneler ile mobile uygulama geliştirilebilir.

Facebook, Instagram, Skype, Uber, Pinterest

Desktop Dev

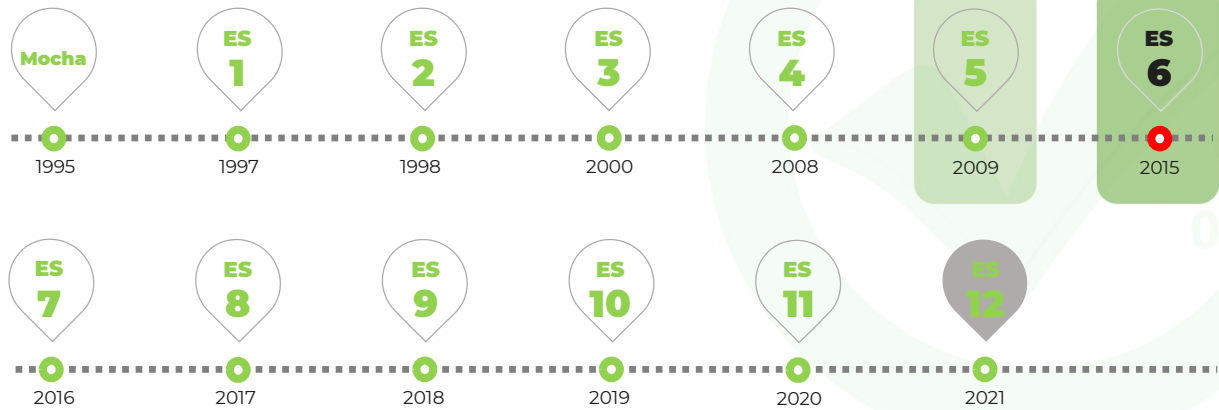


Electron kütüphanesi ile masaüstünde çalışacak uygulamalar geliştirilebilir.

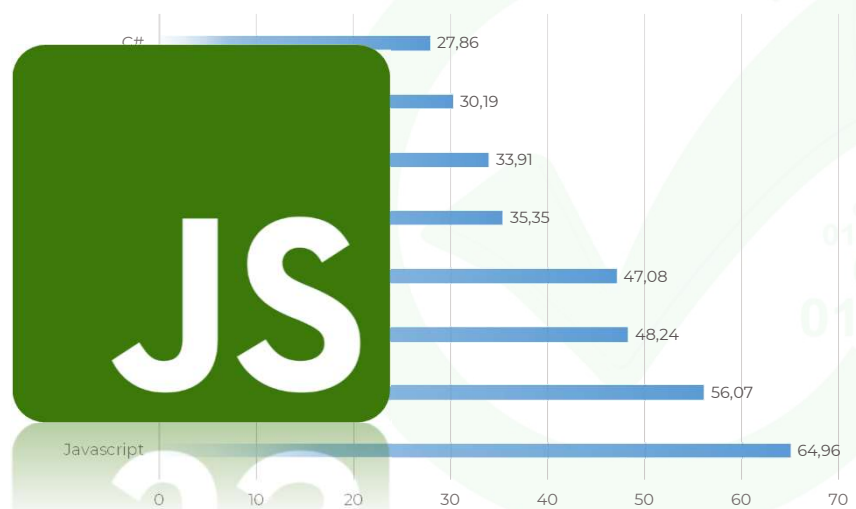
VSCode, Whatsapp, Slack, Skype, Twitch, Teams



Java Script in Gelişimi



Popülerlik

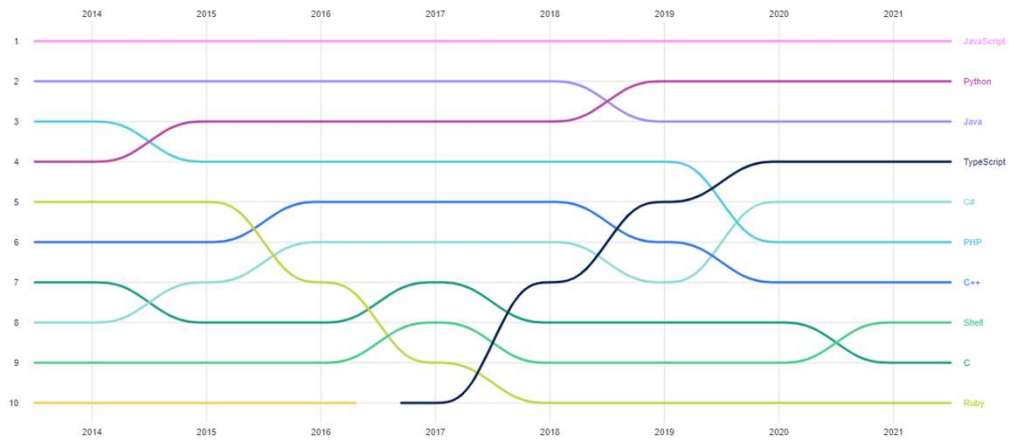


<https://www.statista.com>



Popülerlik

2 milyar civarındaki internet sitesinde %95 oranında Javascript kullanılmaktadır.



Neden popüler?


Kurulum
gerekmez


Object
Oriented


End to end
programming


Library &
Framework


Community



Case Types



camelCase



snake_case



kebab-case



Train-Case



PascalCase



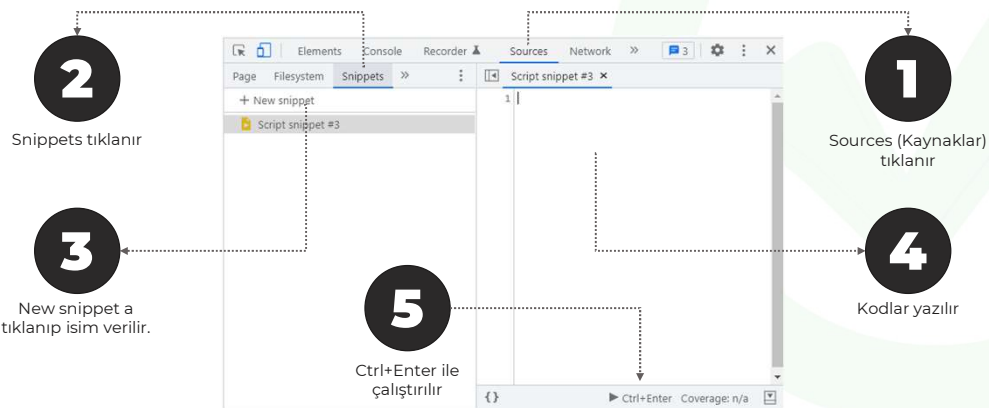
MACRO_CASE



İlk kodlar

Javascript in test amaçlı kullanımı

Boş bir chrome sayfasında sağ tıklayıp inspect (incele) dedikten sonra





ilk kodlar

```
<html>
  <head>

  </head>
  <body>

    <script>
      ...
    </script>
  </body>
</html>
```

- Javascript kodlarını yazmak için script tagi kullanılır.
- Tüm javascript kodları script tagları arasına yazılır
- Script taglarının body kapanmadan hemen önce konulması tavsiye edilir.
- Bu kullanıma internal script denir ve javascript kodları sadece mevcut sayfa için geçerli olur.



ilk kodlar

```
<html>
  <head>

  </head>
  <body>

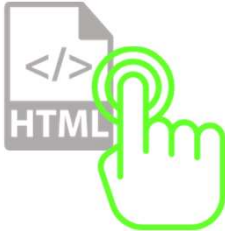
    <script src="assets/js/script.js">
  </body>
</html>
```

- Javascript kodları harici bir dokümana konularak kullanılırsa buna external script denir
- Bu dosyayı kullanan tüm html sayfalarında kullanılabilir hale gelir.



Kodların alıřtırılması

1



Web sayfalarında bulunan Javascript kodları, tarayıcı tarafından alıřtırıldıđı için html dosyasına dosyaya ift tıklayarak amak javascript dosyalarının alıřması için yeterlidir.

2



VSCoDe Live Server eklentisi ile alıřtırılırsa, yapılan deđiřiklikler de anında yansıtılmıř olur. Ve geređe yakın bir ortamda geřiřtirme yapılabilir.



JavaScript

Fundamentals

- › Syntax
- › Console
- › Alert, confirm, prompt
- › Comment
- › Variables and constants
- › Operators



Javascript syntax



- › Case sensitive bir dildir.
- › Satır sonlarına genellikle noktalı virgül (;) konulur.
- › Identifier lar(variables, constants, functions) harf, _ veya \$ ile başlayabilir.
- › Identifier lar **camel case** ile yazılır.



Console

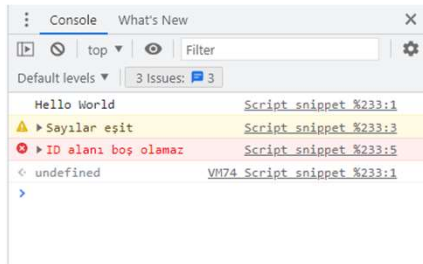
Javascript ile tarayıcıların console bölümüne birşeyler yazdırmak için **console** object kullanılır.

console.log("bla bla bla");



Console

Console object inin farklı kullanımları vardır.



```
console.log("Hello World");
console.warn("Sayılar eşit");
console.error("ID alanı boş olamaz");
```



Alert, Confirm, Prompt

Alert

Bilgilendirme yapmak için kullanılır
alert("Silme yetkiniz yok");

Silme yetkiniz yok.

Tamam

Confirm

Soru sormak için kullanılır.
confirm("Silme istediğinizden emin misiniz?");

Silme istediğinizden emin misiniz?

Tamam

İptal

Prompt

Bilgi almak için kullanılır.
prompt("Adınızı giriniz");

adınızı giriniz

Tamam

İptal



Comment

// Bu bir comment satırıdır.

/*

Bu şekilde birden çok satırı aynı anda comment içine alabilirsiniz.

*/



Variables (Değişkenler), Constants (Sabitler)

CANADA	CAD	0.9512	0.8883
CHINA	CNY	7.3169	6.0910
EURO	EUR	0.6644	0.6100
JAPAN	JPY	109.00	102.00
SINGAPORE	SGD	1.3712	1.2630
HONG KONG	HKD	7.7643	6.4072
NEW ZEALAND	NZD	1.646	1.0875
NEW ZEALAND	NZD	1.2536	2.7818
NEW ZEALAND	NZD	1.2536	2.7818
HONG KONG	HKD	7.7643	6.4072

► Programlama dillerinde ihtiyaç olduğu an ulaşılacak veri tutuculara **değişken** ya da **sabit** adı verilir.



Variables - Constants

Javascript te değişken veya sabit tanımlamak için **3** farklı ifade kullanılır

var

Değişken

ES
6
let

ES
6
const

sabit



Variables - Constants

```
var sayi;  
var Sayi;  
var genelOrtalama;  
const kdv = 1.18;
```

- › case sensitive yani büyük küçük harf duyarlıdır.
- › Sabitler tanımlanırken değerleri de verilmelidir.
- › Değişkenler camelCase olarak tanımlanır.
- › İçinde saklayacağı veriyi anımsatacak bir isim verilir.



Variables - Constants

KURALLAR

- Değişken isimlendirilirken hem harfler hem de sayılar kullanılabilir. Ancak sayılar başa gelmez. Örneğin **sayi1** doğru bir isimlendirmeyken **1sayi** doğru bir isimlendirme değildir.
- Değişken isimlendirilirken alt tire (_ ve \$) kullanılabilir. Ancak boşluk ve diğer özel karakterler (?, %, !, ., + vb.) kullanılmaz. Örneğin **ev adresi** ya da **kimlik%no** gibi değişken isimleri kurallara aykırı olduğundan hataya neden olacaktır.
- Değişken isimlendirilirken özel kullanım için ayrılmış olan **if**, **for**, **true** vb. ifadeler kullanılmamalıdır.
- İngiliz alfabesinde bulunmayan karakterler (**ç,ğ,ı,ö,ş,ü**) kullanılamaz.



Variables - Constants

PRACTISE

✓ders

✗1not

✓gen_ort

✓\$maas

✗ilk sayi

✗son_şekil

✗gun%

✗harf-bir

✓dGunu



Data types (Veri tipleri)

String

var adi="Ali";

Number

var yas=18;
Tam sayı: $\pm 2^{53}-1$
Decimal: $\pm 2^{1024}$

ES
11

BigInt

var x = 100n;

Boolean

var emekli=true;

ES
6

Symbol

var id= Symbol();

Undefined

var isim;

null

var isim = null;

Object

array, object



typeof



var

```
let isim = "Ali";
console.log(isim); .....> Ali
console.log(typeof(isim)); .....> string

let puan = 50;
console.log(puan); .....> 50
console.log(typeof(puan)); .....> number

let emekli = true;
console.log(emekli); .....> true
console.log(typeof(emekli)); .....> boolean
```



var

```
var degisken = "Ali";  
console.log(degisken);  
console.log(typeof(degisken));
```

Ali
String

```
var degisken = 50;  
console.log(degisken);  
console.log(typeof(degisken));
```

50
number

```
var degisken = true;  
console.log(degisken);  
console.log(typeof(degisken));
```

true
boolean



var ile değişken tanımlandığında, aynı isimde başka bir değişken tanımlanmasına izin verir.



const

```
const degisken = "Ali";  
console.log(degisken);  
console.log(typeof(degisken));
```

Ali
string

```
var degisken = 50;  
console.log(degisken);  
console.log(typeof(degisken));
```

Burada
hata verir



const veya let ile değişken tanımlandığında, aynı scope içinde aynı isimde başka bir değişken tanımlanmasına izin **vermez**.



Operatörler



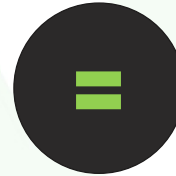
Aritmetik
Operatörler



Karşılaştırma
Operatörleri



Mantıksal
Operatörler



Atama
Operatörler



Aritmetik operatörler

Operatör	Sembol	Örnek
Toplama	+	45+6
Çıkarma	-	35-7
Çarpma	*	25*2
Bölme	/	16/5
Üs Alma	**	5**3
Mod Alma	%	12%5
Bir artırma	++	x++
Bir azaltma	--	y--



Aritmetik operatörler (+)

```
const ekmek = 2;
const yumurta = 30;
const peynir = 40;
const toplamHarcama = ekmek + peynir + yumurta;
console.log("HARCAMA:" + toplamHarcama + " TL");
```

ÇIKTI
HARCAMA: 72 TL

```
const ad = 'Ali';
const soyAd = 'Gel';
console.log(ad + soyAd);
console.log(ad + ' ' + soyAd);
```

ÇIKTI
AliGel
Ali Gel

ÖNEMLİ
+ operatörü ile **string** birleştirme de gerçekleştirilebilir.

```
const x = 5;
const y = "5";
const birlestir = x + y;
console.log(birlestir);
```

ÇIKTI
55



Tip dönüşümleri

```
const para = "100";
console.log(para + 15);
console.log(Number(para) + 15);
```

ÇIKTI
10015
ÇIKTI
115

ÖNEMLİ
Number() fonksiyonu tip çevrimi yapılabilir.

```
const dil = "Javascript";
console.log(Number(dil));
console.log(Number("123abc"));
```

ÇIKTI
NaN
ÇIKTI
NaN

ÖNEMLİ
Number() fonksiyonu harfleri sayıya çeviremeyeceği için NaN (Not a number – Sayı değil) döndürüyor.

```
const para = 5400;
console.log(para+15);
console.log(String(para)+15);
```

ÇIKTI
5415
ÇIKTI
540015

ÖNEMLİ
String() fonksiyonu ile verilen ifadeyi string'e çevirmek mümkündür.



Tip dönüşümleri

```
const s1 = 5;
const s2 = -7;
const isim = "John";
console.log(Boolean(isim));
console.log(Boolean(s1));
console.log(Boolean(s2));
```

ÇIKTI
true
true
true

ÖNEMLİ

0, null, undefined, NaN, ve " " Javascript tarafından **false** olarak kabul edilir.

Diğer değerler Boolean'a çevrildiğinde **true** olarak kabul edilir.

```
const sifir = 0, nal = null;
const tanımsız = undefined;
const bos = "", sayıDegil = NaN;
```

```
console.log(Boolean(sifir), Boolean(nal));
console.log(Boolean(tanımsız), Boolean(bos));
console.log(Boolean(sayıDegil));
```

ÇIKTI
false
false
false



İşlem önceliği

Aritmetik işlemler yapılırken bilgisayar operatör önceliğine göre işlem yapar.



Öncelik seviyesi aynıysa soldaki ifadeyi önce yapar.

() Parantez

****** Üs alma

***** Çarpma

/ Bölme

+ Toplama

- Çıkarma



İşlem önceliği

$$8/2*(2+2)$$
16

$$30 - 3**2 / 3 + 10$$
37

$$16 / 2 * 3 - 2**(4 / 2)$$
20

$$(14 * 2 / 7)**2 / 4 + 5$$
9


String literals

**ES
6**

Metinleri daha dinamik bir şekilde birleştirmek için **string şablonları** (string literals, template literals) kullanabiliriz. Bu birleştirme işlemine de **string interpolation** denir.

```
let isim = "Ali";
let soyisim = 'Gel';
```

Tek tırnak veya çift tırnak

```
let isim = "Ali";
console.log(`Merhaba ${isim}`);
```

string interpolation backtick ile yapılır



Backtick multiline text tanımlamaya da izin verir.



Aritmetik operatörler (-)

```
const yil = 2021;  
const dogumTarihi = 1980;  
const yas = yil - dogumTarihi;  
console.log("YAŞ:" + yas);  
console.log("YAŞ:" + yil - dogumTarihi);
```

ÇIKTI
YAŞ: 41
NaN

ÖNEMLİ

Ekstra **parantez** kullanılmaz ise **string** birleştirme yapmaya çalışır. - den dolayı birleştiremez ve **NaN** döndürür.

NaN = Not a Number (Sayı değil)



Aritmetik operatörler (*, **)

```
const pi = 3;  
const r = 3;  
const alan = pi*r**2;  
const çevre = 2*pi*r;  
console.log(çevre, alan);  
console.log("ÇEVRE:" + çevre, "ALAN:" + alan);
```

ÇIKTI
18 27
ÇEVRE:18 ALAN:27



Aritmetik operatörler (++ , -- , %)

```
let a = 3; let b = ++a; let c = --a;
console.log(a,b,c);
```

ÇIKTI
3 4 3

```
a += 5;
console.log(a);
```

ÇIKTI
8

```
const z = 3;
let k = z++;
console.log(k);
```

HATA
const değişkenin değeri arttırılamaz.

```
const sayi = 123;
console.log("Birler Basamağı:" + sayi%10);
```

ÇIKTI
Birler Basamağı: 3



Karşılaştırma operatörleri

Operatör	Sembol	Örnek
Eşittir	== veya ===	Not==75
Eşit değildir	!= veya !==	Not != 45
Büyüktür	>	Yas > 45
Küçüktür	<	Yas < 18
Büyük veya eşit	>=	Ort >=80
Küçük veya eşit	<=	Ort <=90



Karşılaştırma işlemlerinin sonucu **TRUE** veya **FALSE** olur



Karşılaştırma operatörleri

```
const s1 = 5;
console.log(s1 == 5); // true
console.log(s1 == "5"); // true
console.log(s1 === "5"); // false
console.log(s1 !== 5); // false
console.log(s1 != "5"); // false
console.log(s1 !== "5"); // true
```

```
console.log(s1 > 5); // false
console.log(s1 > "4"); // true
console.log(s1 >= 5); // true
console.log(s1 > "6"); // false
```

ÖNEMLİ:

=== ve !== operatörleri hem veriyi hem de veri tipini kontrol eder.

= ve != operatörleri ise sadece veriyi kontrol ederler.

ÖNEMLİ:

Büyük eşit ve küçük eşit işlemlerinde veri tipi kontrolü yapılmıyor.



Mantıksal operatörler

Operatör	Sembol	Örnek
And	&&	ort > 50 && ort < 70
Or		yas < 16 yas > 70
Not	!	!(yas > 45)

A	B	A && B	A B
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

A	!A
1	0
0	1



Mantıksal operatörler

```
let s2 = true;
let s3 = true;
console.log(s2 && true);      true
console.log(s2 && s3);        true
console.log(s2 && s3 && false); false

s3 = false;
console.log(s2 || s3 || false); true
console.log(s3 || (s2 && s3));  false

s3 = null;
console.log(s2 && s3);         null
console.log(s2 || s3);         true
```

ÖNEMLİ:

0, FALSE, NULL, undefined, "" ve **NaN** dışındaki durumlar **TRUE** kabul edilir.



Mantıksal operatörler

```
s2 = "kuş";
s3 = "kedi";
console.log(s2 || s3);      kuş
console.log(s2 && s3);      kedi

s2 = true;
s3 = false;
console.log(!s2);           false
console.log(!s3);           true

s3 = null;
console.log(!s3);           true
```

ÖNEMLİ:

- **VEYA** işleminde ilk **TRUE** değer bulunması yeterlidir. Diğerlerinin kontrolüne gerek yoktur. Bu yüzden, ilk true olan değişkenin değeri döndürülür. Hiçbiri true değilse en sondaki değer döndürülür.
- **VE** işleminde ilk **FALSE** değer bulunması yeterlidir. Diğerlerinin kontrolüne gerek yoktur. Bu sebeple ilk false olan değişkenin değeri döndürülür. Hiçbiri false değilse en sondaki değer döndürülür.



Mantıksal Operatörler

not1	50
not2	60
not3	70

T	not1 > 30	&&	not1 < 70		
F	not2 >= 60	&&	not3 != 70		
T	not3 < 100		not2 > 80		
F	not1 + not2 < not3		not1 < 50		
T	not1 < 30		not2 > 50	&&	not3 == 70



Atama operatörleri

Operatör

Örnek

Açıklama

=	x = y	Sağdaki değişkenin değerini soldakine kopyalar.
+=	x += 1	x = x+1 işlemi gerçekleştirir.
-=	x -= 2	x = x-2 işlemi gerçekleştirir.
*=	x *= 3	x = x*3 işlemi gerçekleştirir.
/=	x /= 4	x = x/4 işlemi gerçekleştirir.
**=	x **= 2	x = x ² işlemi gerçekleştirir.
%=	x %= 3	x = x mod 3 işlemi gerçekleştirir.
&=	x &= y	x = x VE y işlemi gerçekleştirir.
=	x = y	x = x VEYA y işlemi gerçekleştirir.



Atama operatörleri

```
let x = 22;
```

```
let y = 5;
```

```
x += 4;
```

```
x=x+4
```

```
x=22+4
```

```
x=26
```

```
x /= 2;
```

```
x=x/2
```

```
x=26/2
```

```
x=13
```

```
x %= y;
```

```
x=x%y
```

```
x=13%5
```

```
x=3
```

```
y = x;
```

```
y=x
```

```
y=3
```

```
x *= y;
```

```
x=x*y
```

```
x=3*3
```

```
x=9
```

```
y **= 2;
```

```
y=y2
```

```
x=32
```

```
y=9
```

```
console.log(x-y);
```

```
0
```



JavaScript

Temel DOM İşlemleri

- › document object
- › querySelector
- › innerText
- › innerHtml
- › value
- › onclick



Document object

Bir web sayfasındaki diğer tüm nesnelerin sahibi olan ana nesneye document object denir. Document nesnesi web sayfamızı temsil eder. Eğer sayfa içindeki bir nesneye erişmek istenilirse **document** nesnesi kullanılır.

```
document.querySelector(selector);
```

Selectorle eşleşen ilk elemanı seçer

css selectorleri kullanılır. (menü li)



Document object

```
document.querySelectorAll(selector);
```

Selectorle eşleşen tüm elemanları seçer ve bir dizi içinde tutar.

css selectorleri kullanılır. (menü li)



Document object

innerText

Bir elementin içindeki yazıyı okumak veya değiştirmek için kullanılır

innerHTML

Bir elementin içindeki HTML'i okumak veya değiştirmek için kullanılır

classList

Bir elementin sahip olduğu class ları manipüle etmek için kullanılır

add

Elemente bir class eklemek için kullanılır

remove

Elementin sahip olduğu bir class ı kaldırmak için kullanılır



querySelector

```
<h1 class="name">Ali Gel</h1>  
<p class="mesaj"></p>
```

```
let isim = document.querySelector('.name').innerText;  
document.querySelector('.mesaj').innerText = `Merhaba ${isim}`;  
  
document.querySelector('.mesaj').innerHTML = `<b>Merhaba ${isim}</b>`;
```



querySelector

```
<h1 class="name blue">Ali Gel</h1>  
<p class="mesaj"></p>
```

```
document.querySelector('.mesaj').classList.add('red');  
document.querySelector('.mesaj').classList.remove('blue');
```



querySelector

value

Bir form elemanının değerini okumak ya da değiştirmek için kullanılır

```
<input type="text" id="name">  
<input type="text" id="mesaj">
```

```
let isim = document.querySelector('#name').value;  
document.querySelector('#mesaj').value = `Merhaba ${isim}`;
```



onclick

```
<p onclick="sayHello()">Merhaba Dünya</p>
```

```
function sayHello(){  
  alert("Hello World!");  
}
```



DOM

Textbox lardan alınan iki sayıyı, butona basılınca toplayan ve sonucu bir label içinde gösteren sayfayı yapınız

Sonuç:

PRACTISE



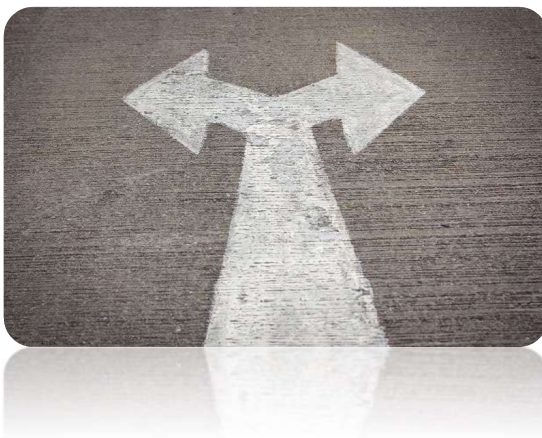
JavaScript

Conditional Statements

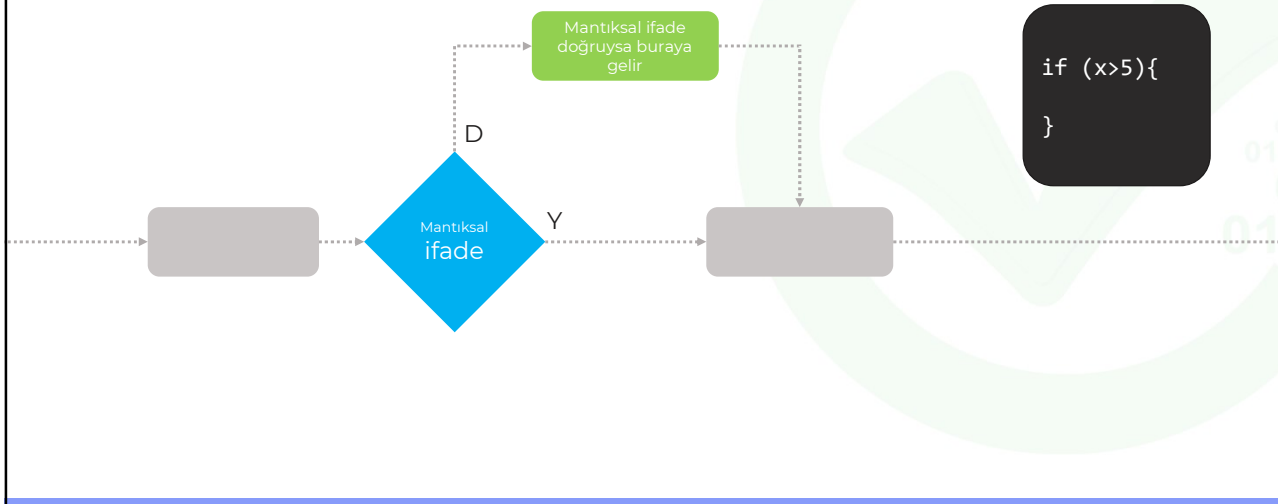
- › if
- › if-else
- › if elseif
- › switch
- › ternary
- › || and &&



Conditional Statements



- › Programlamada program akışını çeşitlendirmek için kullanılır.
- › Javascript te bunun için if veya switch blokları kullanılır.

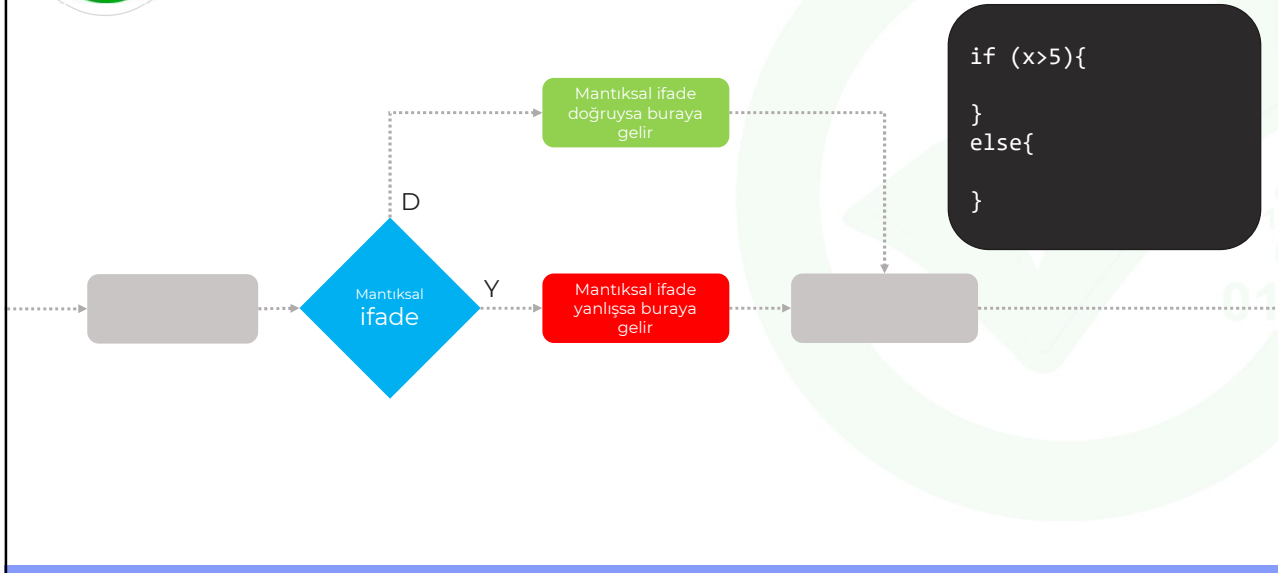
**if****if**

Bir Textbox tan alınan fiyat bilgisi 500 den büyükse %10 indirim uygulayıp sonucu yazan, değilse doğrudan sonucu yazan programı yapınız.

PRACTISE



if-else



if-else

Bir Textbox tan alınan fiyat bilgisi 500 den büyükse %10 indirim uygulayıp sonucu yazan, değilse %5 indirip uygulayıp sonucu yazan programı yapınız.

PRACTISE



DOM

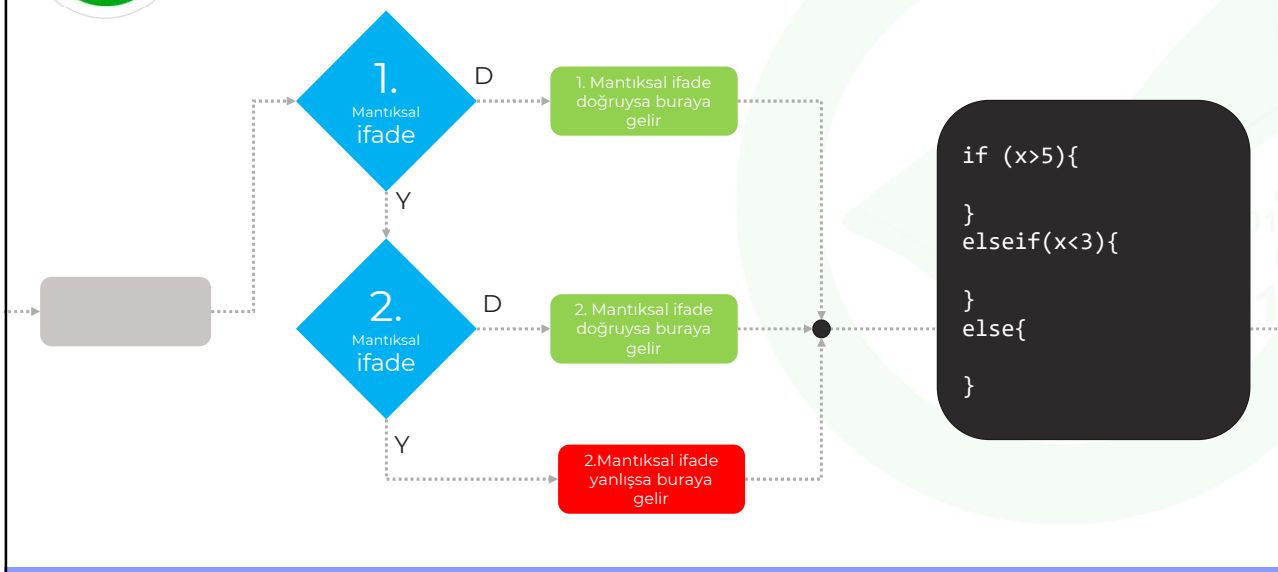
PRACTISE

Dark-light mode switcher uygulaması yapınız.

	Dark Mode	Light Mode
Page background color	black	White
Page text color	white	Black
Button background color	white	Black
Button text color	black	White



if-elseif-else





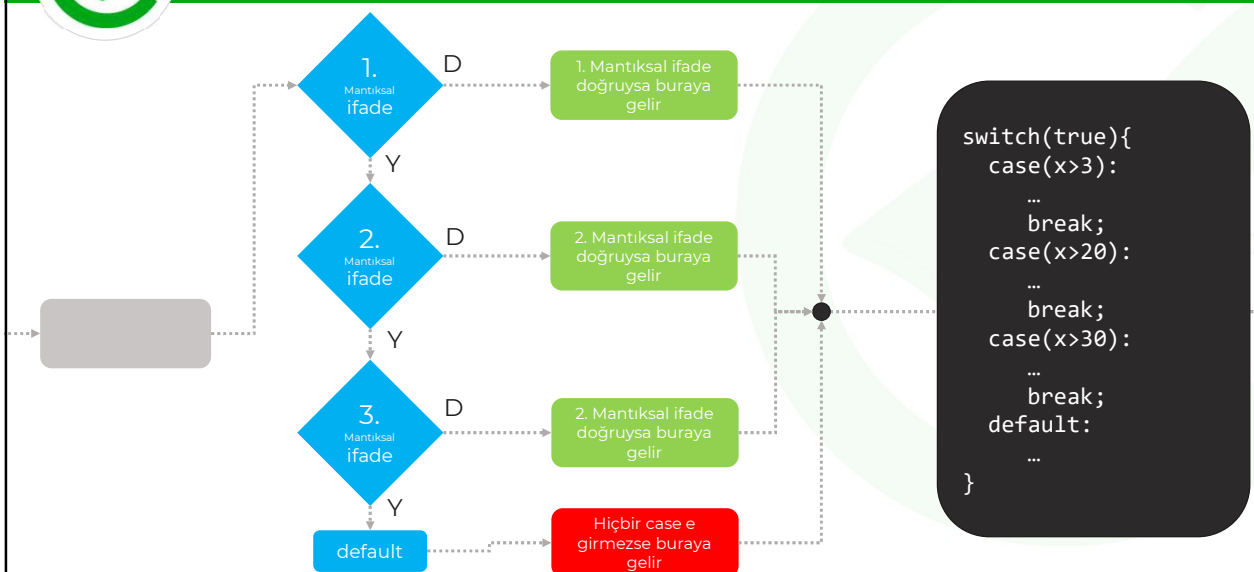
if-elseif-else

PRACTISE

Bir Textbox tan alınan fiyat bilgisi 500 den büyükse %10 indirim uygulayıp sonucu yazan, 300 den büyükse %7 indirip uygulayıp sonucu yazan, değilse %5 indirip uygulayıp sonucu yazan programı yapınız.



switch





switch

PRACTISE

Bir Textbox tan alınan fiyat bilgisi
1500 den büyükse %15,
1000 den büyükse %13,
700 den büyükse %10,
500 den büyükse %7,
300 den büyükse %5,
Diğer durumlarda %2 indirim uygulayıp sonucu yazan programı yapınız.



ternary

if-else bloğunun tek satırda basit bir şekilde yazılmasıdır

mantıksal_ifade ? doğruysa : yanlışsa

```
let maas = 15000;  
console.log( maas>13000 ? "Normal" : "Az" );
```



ternary

PRACTISE

☐ Emekli
☒ Çalışan
Maaş=13500

- › Şekildeki gibi form oluşturunuz. Çalışan seçildiğinde tam, emekli ise %10 kesinti uygulayarak maaşı gösteriniz. Bunu ternary kullanarak yapınız.



|| ve &&

if bloğunun tek satırda basit bir şekilde yazılmasıdır

mantıksal_ifade **&&** doğruysa
mantıksal_ifade **||** yanlışsa

```
var x = 15  
x == 10 && alert("x is 10");  
x == 10 || alert("x is not 10");
```



Hesap Makinesi

HOMEWORK

İki textbox, select, button ve label oluşturunuz. Textbox lara girilen sayıları select ten seçilen işlem türüne göre, hesapla butonuna basıldığında işleme tabi tutup sonucu label da gösteren programı yazınız.



JavaScript

Basic Loops

- › for
- › while
- › do-while



Döngüler

Tekrarlanan işlemleri daha az kod yazarak gerçekleştirmek için döngüler kullanılır.

for

while

do-while



for

```
for(var i=0; i<10; i++){
```

.

.

```
}
```

Döngünün
başlangıç
sayısı

Doğru
olduğu
süreçe
döngü tekrar
eder

Artış
miktarı

- › En temel döngü türüdür.
- › Tekrar ettirilecek kodun baştan kaç defa tekrar ettirileceği biliniyorsa bu tür döngüler kullanılabilir.

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

**for****PRACTISE**

İki textbox tan alınan sayılar arasında 7 ile bölünebilen sayıların adedini bulan ve sonucu bir h1 içinde gösteren programı yazınız.

**for****PRACTISE**

- Notebook 1500\$
- Monitor 500\$
- Klavye 10\$
- Mouse 5\$

Bir ul listesinde ürün isimleri ve fiyat bilgileri bulunmaktadır. Textbox a girilen indirim oranı kadar listeye indirim uygulayan programı yazınız.



while

```
while(a<50){  
    ...  
}
```

Doğru
olduğu
sürece
döngü tekrar
eder

- › Döngünün ne kadar döneceği döngü bloğunun içindeki kodlara bağlı ise bu durumda while döngüleri kullanılabilir.



do-while

```
do{  
    ...  
}  
while(a<50)
```

Doğru
olduğu
sürece
döngü tekrar
eder

- › Do while da kontrol sonda olduğu için döngü içindeki kodlar en az bir kere icra edilir.



do-while

PRACTISE

İki textbox tan alınan sayılar sırayla toplanmaya başlandığında, genel toplamı 2000 i ulaştıran kaç adet sayı olduğunu bulan ve sonucu alert ile gösteren programı yapınız.



JavaScript

Functions

- › Fonksiyon nedir?
- › Nasıl çalışır?
- › Fonksiyon tanımlama
- › Scope



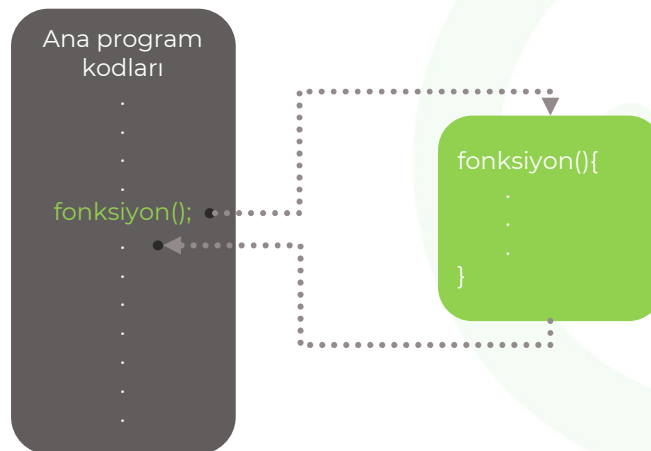
Fonksiyon nedir?

Fonksiyon, programda tekrar tekrar çalıştırılabilecek alt programlara denir.

- Kod tekrarlarının önüne geçer
- Problemler küçük parçalara ayrılarak daha kolay çözülür.
- Parçalara ayrılan sorun, daha fazla insan tarafından aynı anda çözülür.
- Hata ayıklama daha kolay olur



Nasıl çalışır





Fonksiyon tanımlama

Geleneksel yöntem

```
function fonksiyonIsmi(param1, param2, ...){  
  // işlemler  
  // ...  
  return donenDeger;  
}
```

Fonksiyona dışarıdan
gelen parametreler

Fonksiyonun çağrıldığı
yere döndürdüğü değer

```
var x = fonksiyonIsmi(paramValue1, paramValue2, ...);
```

Fonksiyonun çağırılması



Fonksiyon tanımlama

Yaş hesaplayan bir fonksiyon yazınız.

```
function yasHasapla(){  
  console.log(`Benim yaşıım ${2022 - 1990}`);  
}  
  
yasHasapla();  
yasHasapla();
```

Bu fonksiyon hep aynı yaşı
gösteriyor. Farklı doğum
tarihlerine göre daha
modüler hale getirilebilir
mi?



Fonksiyon tanımlama

Yaş hesaplayan bir fonksiyon yazınız.

```
function yasHasapla(dogumTarihi){  
    console.log(`Benim yaşıım ${2022 - dogumTarihi}`);  
}  
  
yasHasapla(1980);  
yasHasapla(2001);
```

Eğer yaş bilgisi ana programda kullanılmak istenirse ne yapılabilir?



Fonksiyon tanımlama

Yaş hesaplayan bir fonksiyon yazınız.

```
function yasHasapla(dogumTarihi){  
    return 2022 - dogumTarihi;  
}  
  
var yas = yasHasapla(1980);  
if(yas >= 65){  
    ...  
}
```



Fonksiyon

PRACTISE

Bir sayının asal olup olmadığını bulan fonksiyonu yazınız.



Fonksiyon tanımlama

Expression yöntemi

```
const yasHasapla = function(dogumTarihi){  
  return 2022 - dogumTarihi;  
}  
  
yasHasapla();
```



Fonksiyon tanımlama

ES
6

Arrow function yöntemi

```
const yasHasapla = (dogumTarihi) => 2022-dogumTarihi;
```

```
let yas = yasHasapla(2000);
```

Eğer fonksiyon içeriği tek satır ise {} kullanmadan doğrudan değer döndürülebilir.

```
const yasHasapla = (dogumTarihi) => {  
  if(!dogumTarihi) return;  
  return 2022-dogumTarihi;  
}
```

```
let yas = yasHasapla(2000);
```

Eğer fonksiyon içeriği birden fazla satır ise {} ve return kullanılmalıdır.



Fonksiyon tanımlama

Arrow function

```
const toplama = (a, b) => a+b;
```

Expression

```
const toplama = function(a, b){  
  return a+b;  
}
```

Geleneksel

```
function toplama(a, b){  
  return a+b;  
}
```



Fonksiyon

PRACTISE

Textbox a girilen sayının, sayı olup olmadığını kontrol ettikten sonra, faktöriyelini hesaplayıp sonucu döndüren fonksiyonu arrow function yöntemi ile yapınız.



Scope

Değişken ve sabitlerin yaşam alanına scope denir

function scope

Fonksiyon içinde tanımlanan değişkenler sadece fonksiyon içinde geçerlidir.

global scope

Her yerden erişilebilen değişkenler, en tepede tanımlanır.

block scope

Sadece tanımlandığı blok içinde (if, for, while...) geçerlidir.



Function Scope

```
const fonk1 = function () {
  let sayi1 = 22;
  console.log(sayi1);
};

fonk1();
console.log(++sayi1);
```

sayi1 değişkeni fonk1 içinde tanımlandığı için, sadece fonk1 fonksiyonu içinde geçerlidir. Fonksiyon dışından erişilmeye çalışıldığında hata alınır.

Uncaught ReferenceError: sayi1 is not defined
at Script snippet %235:6:9



Global Scope

```
let sayi = 5;
const fonk = function () {
  sayi = 10;
  console.log(`Fonk. İçi: ${sayi}`);
}

fonk();
console.log(`Fonk. Dışı: ${++sayi}`);
```

sayi değişkeni fonk dışında tanımlandığı için, her yerden erişilebilir. Örnekte hem fonk isimli fonksiyondan hem de dışındaki kodlar üzerinden erişilebilmektedir.

Fonk. İçi: 10

Fonk. Dışı: 11



Global vs Function Scope

```
let sayi = 3;  
const fonk = function () {  
  let sayi = 7;  
  console.log(`Fonk. İçi: ${sayi}`);  
};  
  
fonk();  
console.log(`Fonk. Dışı: ${++sayi}`);
```

Fonk. İçi: 7
Fonk. Dışı: 4

Hem global hem de function scope da aynı isimde değişken tanımlandığında bunları farklı değişkenler olarak algılanır.



Block Scope

ES
6

```
const fonk = function (sayi) {  
  if (sayi < 0){  
    let negatif = true;  
  }  
  console.log(negatif);  
};  
fonk(-4);
```

Uncaught ReferenceError: negatif is not defined
at fonk (a.html:15:21)
at a.html:17:7

Block scope değişkenler sadece bulundukları {} blok içinde geçerlidir. Blok dışından erişim yapılamaz. Block scope sadece let ve const için geçerlidir.



var vs let

var ile **let** arasındaki en önemli fark, scope farklılığıdır. **var** function scope, **let** block scope olarak davranır.

```
function test(){
  var a = "Merhaba";
  let b = "Dünya";
  if(true){
    var x = "Ankara";
    let y = "İzmir";
  }
  console.log(a, b, x);
  console.log(y);
}
```

Merhaba Dünya Ankara

✖ ▶ Uncaught ReferenceError: y is not defined



Scope

«Oyuna Başla» butonuna basıldığında 1-100 arasında rasgele sayı tutulsun. Textbox içine tahmini sayı girilip «tahmin et» butonuna basıldığında tutulan sayıyla girilen sayıyı kontrol edip doğru tahmin olup olmadığını ve büyüklük küçüklük durumunu gösteren programı yapınız.

Yanlış tahmin! Daha büyük bir sayı giriniz.

PRACTISE

İpucu

```
let sayi = Math.floor(Math.random() * (max - min + 1)) + min
```



JavaScript

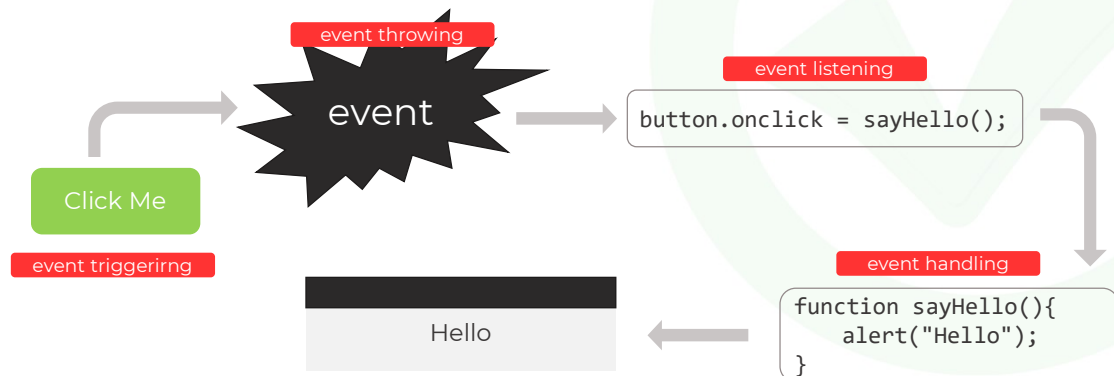
Event Handlers

- › Event handler nedir?
- › Nasıl tanımlanır?
- › Handler çeşitleri



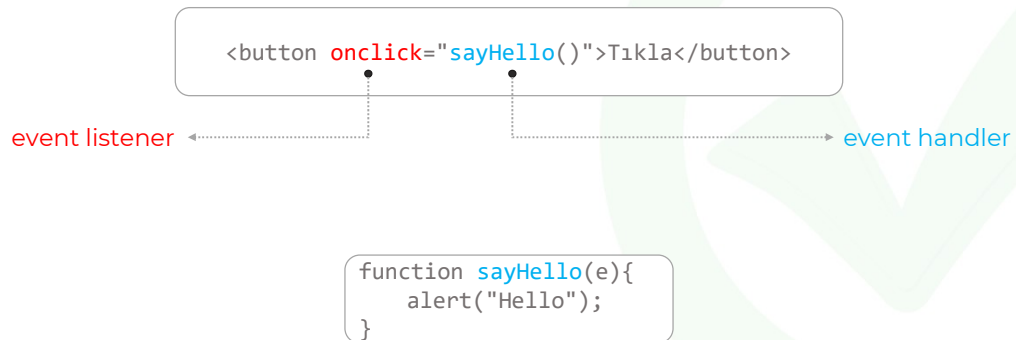
Event handler

Bir programda olabilecek olayların yakalanması ve olay gerçekleştiğinde istenilen kodların çalıştırılması işlemine **event handling**, bu işlemi yapan kodlara da **event handler** denir.

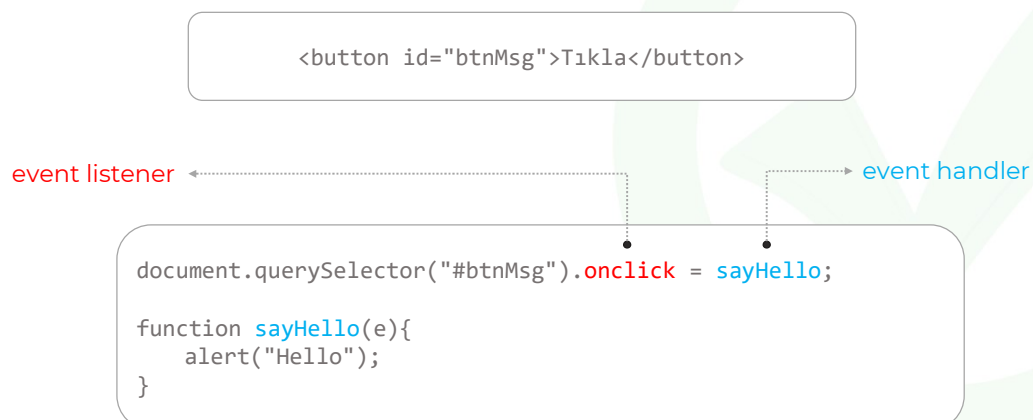




Tanımlama (1.Yöntem)



Tanımlama (2.Yöntem)





Tanımlama (3.Yöntem)

```
<button id="btnMsg">Tıkla</button>
```

event listener

event handler

```
document.querySelector("#btnMsg").onclick = function(e){  
    alert("Hello");  
}
```

```
document.querySelector("#btnMsg").onclick = (e) => {  
    alert("Hello");  
}
```

arrow function



Tanımlama (4.Yöntem)

```
<button id="btnMsg">Tıkla</button>
```

event listener

event handler

```
document.querySelector("#btnMsg").addEventListener("click", sayHello);  
  
function sayHello(e){  
    alert("Hello");  
}
```



Tanımlama (5.Yöntem)

```
<button id="btnMsg">Tıkla</button>
```

event listener

event handler

```
document.querySelector("#btnMsg").addEventListener("click", function(e){
    alert("Hello");
});
```

```
document.querySelector("#btnMsg").addEventListener("click", (e) => {
    alert("Hello");
});
```

arrow function



Function

Butona basıldığında, iki textbox içinde 100 lük sistemde girilen notları kontrol edip, ortalamasını bulan ve bunu harf sistemine çeviren fonksiyonu yazınız.

PRACTISE

Aralık	Not
[90-100]	A
[80-90)	B
[70-80)	C
[50-70)	D
[0-50)	F



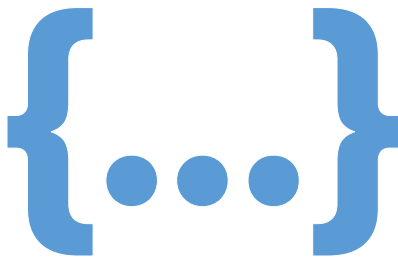
JavaScript

Objects & Arrays

- › Object nedir?
- › Object tanımlama
- › Array nedir?
- › Array tanımlama
- › Array of objects
- › Array Methods
 - › for-in
 - › forEach
 - › map
 - › filter
 - › reduce
 - › every
 - › some



Object nedir?



- Programımızda bir **araba** ya ait çeşitli özellikleri saklamak istiyoruz. Bu durumda, değişken kullanırsak;

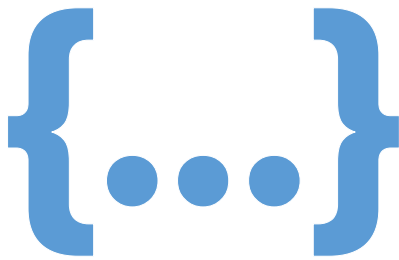
```
const marka = 'Mercedes';  
const model = 'S500';  
const renk = 'Bej';  
const vites = 'Otomatik';
```

Eğer birden fazla arabaya ait bilgiler saklanacaksa?

Object yapıları kullanılarak bir nesneye ait özellikler ve değerleri gruplandırılabilir.



Object nedir?



Javascript te basit object tanımı {} ile yapılır.

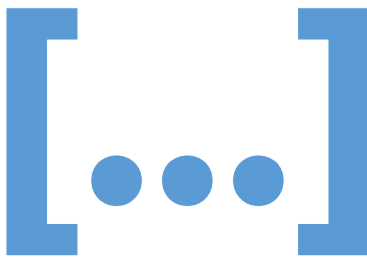
```
const araba = {  
  marka: 'Mercedes',  
  model: 'S500',  
  renk: 'Bej',  
  vites: 'Otomatik'  
}
```

```
console.log(araba);
```

```
console.log(araba.marka);
```



Array nedir?



- Programımızda **araba** isimlerini saklamak istiyoruz. Bu durumda, değişken kullanırsak;

```
const araba1 = 'Mercedes';  
const araba2 = 'TOFAŞ';  
const araba3 = 'Anadol';  
const araba4 = 'Ferrari';
```

**Saklanacak
yüzlerce araba
ismi olsaydı?**

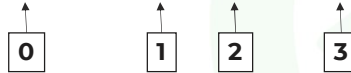
Diziler, bir **veri yapısı** (data structure) olup bir veya çok daha fazla veriyi saklamak için kullanılır.



Array tanımlama

Javascript te basit dizi tanımlı [] ile yapılır.

```
let kurslar = ["Javascript", "HTML", "CSS", "Bootstrap"];
```



Dizi tanımlama için bir diğer yöntem

```
let kurslar = new Array("Javascript", "HTML", "CSS", "Bootstrap");
```



Dizilerden veri okumak

```
var kurslar = ["Javascript", "HTML", "CSS", "Bootstrap"];
```

Dizi elemanlarına erişim

```
kurslar[1];
```

Dizi uzunluğu

```
kurslar.length
```

```
console.log(kurslar[1]);
```

```
console.log(kurslar[kurslar.length-1]);
```

Dizi elemanları **sıralı** olarak belleğe kaydedilirler.

Sıra numaraları **0** ile başlar ve dizinin **eleman sayısının bir eksiğine (length-1)** kadar devam eder.



Dizilere veri yazmak

```
const kurslar = ["Javascript", "HTML", "CSS", "Bootstrap"];
```

```
kurslar[1] = "React";  
kurslar[0] = "Java";
```

```
console.log(kurslar);
```

```
kurslar = ["Ali", "Veli"];
```



```
✖ ▶ Uncaught TypeError: Assignment to constant variable.  
at main.js:23
```

Dizi **const** ile tanımlanmış olsa da dizinin elemanlarını değiştirebildik. Çünkü **non-primitive** veri türlerinin içerikleri değiştirilebilir. Ancak, tamamen bir başka dizi ile değiştirilemez.



Diziler

PRACTISE

- › Bir dizi içindeki en büyük sayıyı bulan programı yazınız.

```
var sayilar =  
[12, 56, 14, 67, 89, 33, 22];
```



Diziler

PRACTISE

```
const fiyatlar = [123, 5666, 126, 67];
```

Şeklindeki dizi elemanlarının değerini %20 artıran fonksiyonu yazınız.



Array of Objects

```
const arabalar = [{  
  marka: 'Mercedes',  
  model: 'S500',  
  renk: 'Bej',  
  vites: 'Otomatik'  
},  
{  
  marka: 'BMW',  
  model: 'i8',  
  renk: 'Mavi',  
  vites: 'Otomatik'  
},  
...  
]
```

Elemanları object olan dizilere array of object denir.



Array Methods

Dizi elemanlarına erişmek, bunları manipüle etmek veya iterasyon yapmak için çeşitli hazır methodlar bulunmaktadır

mutator methods

Dizi içeriğinin değiştirilmesini sağlarlar

access methods

Dizi elemanlarına erişmek için kullanılırlar

iterator methods

Dizi elemanları döngü yoluyla dolaşmak için kullanılır



Mutator methods

Method	Açıklama
push	Dizinin sonuna bir veya daha fazla eleman ekler ve dizinin yeni eleman sayısını döndürür
pop	Dizinin son elemanını diziden siler ve bu elemanı döndürür
shift	Dizinin ilk elemanını diziden siler ve bu elemanı döndürür
unshift	Dizinin ilk indeksine yeni bir eleman ekler ve dizinin yeni eleman sayısını döndürür
sort	Diziyi yükselen sırada sıralar
reverse	Dizinin elemanlarını ters sıralar
splice	Dizi'nin içeriklerini, yeni öğeler ekleyerek, mevcut öğeleri silerek günceller.



Mutator methodlar

```
const meyveler = ["Elma", "Armut", "Muz", "Kivi"];
meyveler.pop(); Kivi silindi
meyveler.push('Ananas'); Sona Ananas eklendi
meyveler.shift(); Elma silindi
meyveler.unshift('Çilek'); Başa Çilek eklendi
meyveler.reverse(); Tersine çevrildi
meyveler.sort(); Harfe göre sıralandı
meyveler.splice(1,0,'Kiraz'); 1.İndex e Kiraz eklendi
meyveler.splice(3, 1 ,'Kayısı'); 3.İndex e Kayısı eklendi, Muz silindi
```



Access methods

Method	Açıklama
concat	Dizi ile bir başka diziyi veya değeri birleştirerek yeni bir dizi döndürür.
includes	Dizinin belirtilen bir elemanı içerip içermediğine bakar. Eğer içeriyorsa true, içermiyorsa false döndürür.
indexOf	Belirtilen elemanın dizide ilk görüldüğü indeks numarasını döndürür.
lastIndexOf	Belirtilen elemanın dizide görüldüğü en son indeks numarasını döndürür. Bulunmazsa -1 döndürür.
join	Bir dizi içerisinde yer alan bütün elemanları birleştirerek String bir ifade olarak geri döndürür.
toString	Dizinin içerisindeki elemanları tek bir String olarak döndürür.
slice	Bir Dizinin elemanlarını, belirtilen başlangıç ve bitiş indeksine göre kopyasını oluşturarak ve döndürür.



Access methods

```
const hayvanlar = ['fil', 'kuş', 'deve', 'fare', 'kedi']; hayvanlar.includes('fil');
hayvanlar.includes(3); true
hayvanlar.includes("at"); false
false

console.log(hayvanlar.join());
console.log(hayvanlar.join("")); fil,kuş,deve,fare,kedi
console.log(hayvanlar.join("-")); filkuşdevefarekedi
Fil-kuş-deve-fare-kedi

console.log(hayvanlar.slice(2));
console.log(hayvanlar.slice(2, 4)); ["deve", "fare", "kedi"]
console.log(hayvanlar.slice(1, 5)); ["deve", "fare"]
["kuş", "deve", "fare", "kedi"]

console.log(hayvanlar.toString());
fil,kuş,deve,fare,kedi
```



Access methods

```
const hayvanlar = ['fil', 'kuş', 'deve', 'kuş', 'kedi', 'kuş'];

console.log(hayvanlar.indexOf('kuş')); 1
console.log(hayvanlar.lastIndexOf('kuş')); 5
console.log(hayvanlar.indexOf('at')); -1

const harfler = ['a', 'b', 'c'];
const rakamlar = [1, 2, 3];
const birlesik = harfler.concat(rakamlar, 4, [5,6]);
console.log(birlesik); ['a', 'b', 'c', 1, 2, 3, 4, 5, 6];
```



Iteration - for

ÖRNEK:

Bir dizideki sayıların toplamını hesaplayan uygulamayı For döngüsü ile yazınız.

```
const rakamlar = [-5, 15, 22, -4, 45, 78, -25];  
  
let toplam = 0;  
  
for (let i = 0 ; i < rakamlar.length ; i++) {  
    toplam += rakamlar[i];  
}  
  
console.log(toplam);
```



Iteration - for

Bir dizideki pozitif ve negatif sayıların toplamını hesaplayıp, bu toplamların farkını bulan uygulamayı for döngüsü ile yazınız.

hesapla adında bir fonksiyonda hesaplamayı yapınız.

PRACTISE



Iteration – for in

For döngüsünün kısaltılmış halidir. (sayaç ve koşul kullanmaya gerek yok.) Özellikle **dizi** ve **nesnelerin iterasyonu** için geliştirilmiştir. Değişken içine otomatik olarak elemanın indis numarası gelir.

```
for ( degisken in diziAdi) {  
  // Döngü içi  
}
```



Iteration – for in

```
const adlar = ["Ahmet", "Can", "Mustafa", "Ayşe", "Elif"];  
const soyAdlar= ["Öztürk", "Yılmaz", "Arı", "Çalı", "Yazı"];  
  
const birlestir = (x,y) => {  
  let adVeSoyadlar = [];  
  for (let i in x) {  
    adVeSoyadlar[i] = `${x[i]} ${y[i]}`;  
  }  
  return adVeSoyadlar;  
};  
  
console.log(birlestir(adlar, soyAdlar));
```

ÖRNEK:

İki ayrı dizideki eşleşen indis elemanları birleştirerek ayrı bir diziye saklayan uygulamayı **FOR IN** ile yazınız.



Iteration – for of

**ES
6**

FOR OF, **bir çok veri yapısı** üzerinde çalışabildiği için **FOR IN'e** göre daha geniş kullanım alanına sahiptir. Diziler, Stringler v.b bir çok veri yapısında kullanılabilir. Değişken içine otomatik olarak elemanın değeri gelir.

```
for ( degisken of diziAdi) {  
    // Döngü içi  
}
```



Iteration – for of

```
let arabalar = ["BMW", "Volvo", "Mini"];  
let yazi = "";  
  
for (let x of arabalar) {  
    yazi += x + " ";  
}  
console.log(yazi);
```

ÖRNEK:

Dizideki elemanları birleştirerek tek bir **String** haline getiren uygulamayı **FOR OF** ile yazınız.



For - of

PRACTISE

Harici ülke listesi datasını alarak bir select içinde ülkeleri gösteriniz.



Iteration – for each

FOR OF döngüsüne benzemektedir. Sadece diziler üzerinde çalışır. For of tan sonra popülerliğini yitirmeye başlamıştır.

```
dizi.forEach( (item, index)=> {  
  // Döngü içi  
});
```



Iteration – for each

```
const dizi = [-5, 24, -13, 7];  
const yeniDizi = [];  
  
dizi.forEach((item, index) => { yeniDizi[index] =  
    item * 5;  
});  
console.log(yeniDizi);
```

ÖRNEK:

Belirtilen dizinin her bir elemanının 5 katını alarak ayrı bir dizide saklayan uygulamayı **forEach()** metodu ile yazınız.



Iteration – map

Mevcut bir dizi içinde iterasyon yapıp, orijinal dizinin kopyası üzerinde elemanlarının değerlerini değiştirmeyi sağlar. Değişiklik yapılmış diziyi geri verir. Map methodu orijinal diziyi değiştirmez.

```
dizi.map( (item, index) => item * 5);
```



Iteration – map

```
const isimler = ["Mustafa", "Murat", "Ahmet", "Mustafa", "Ayşe",  
"canan"];  
  
const buyukIsimler = isimler.map((x) => x.toUpperCase());  
  
console.log(buyukIsimler);
```

ÖRNEK:

Bir dizideki tüm isimleri **BÜYÜK** harfe dönüştüren uygulamayı yazınız.



Iteration – map

tlFiyatlar dizisindeki fiyatların Euro ve dolar karşılıklarını hesaplatarak yeni dizilere kaydediniz.

```
const euro = 9.68;  
const dolar = 8.1;  
const tlFiyatlar = [100, 150, 100, 50, 80];
```

PRACTISE



Iteration – for map

PRACTISE

ÖRNEK:

tlFiyatlar dizidekisindeki ürünlere fiyatı 100 TL den fazla olanlara %10 zam, 100 TL den az olanlara ise %15 zam yapılmak isteniyor.

Ayrıca, zamlı olan yeni değerleri yeni bir diziye saklamak istiyoruz.



Iteration – filter

Mevcut bir dizi içinde iterasyon yapıp, orijinal dizinin kopyası üzerinde filtreleme yapar. Filtrelenmiş diziyi geri verir. Filter methodu orijinal diziyi değiştirmez.

```
dizi.filter( (item, index) => item > 5);
```



Iteration – filter

ÖRNEK:

Koordinatlar dizisindeki negatif koordinatları alıp yeni bir diziye saklayan uygulamayı **filter()** ile yapınız.

```
const koordinatlar = [-100, 150, -32, 43, -20];  
const negatifKoordinatlar = koordinatlar.filter((x) => x < 0);  
console.log(negatifKoordinatlar);
```



Iteration – filter

```
const bireyler = ["Mustafa", "Murat", "Ahmet", "mustafa", "Ayşe",  
"Canan"];  
  
const filtrele = function (harf) {  
  const h = harf.toUpperCase();  
  const filtrelenmiş = bireyler.filter((t) => t.startsWith(h));  
  return filtrelenmiş;  
};  
  
console.log(filtrele("m"));  
console.log(filtrele("A"));
```

ÖRNEK:

Bireyler dizisindeki kişilerden adı "**Belirtilen**" harf ile başlayanları seçerek ayrı bir diziye saklayan uygulamayı yazınız.



Iteration – filter

ÖRNEK:

Koordinatlar dizisindeki negatif koordinatları seçerek bunları pozitifte çevirip alt alta konsola bastıran uygulamayı yazınız.

```
const koordinatlar = [-100, 150, -32, 43, -20];  
koordinatlar.filter(x => x < 0)  
  .map(t => t * -1)  
  .forEach(y => console.log(y));
```

İterasyon methodlarının
peşpeşe kullanılmasına
pipeline denir



Filter

Tablo içine yerleştirilen ülkelerden biri seçildiğinde seçilen ülkeye ait bilgileri gösteren uygulamayı filter ile yapınız.

PRACTISE



Iteration – reduce

Mevcut bir dizi içinde iterasyon yapıp, yapılan işlemlere göre tek bir hesaplanmış değer döndürür.

```
dizi.reduce( (total, x) => total + x, 0 );
```

Her bir işlem sonucunu saklayacak olan değer

Her bir dizi elamanı

İlk değer



Iteration – reduce

```
const koordinatlar = [-100, 150, -32, 43, -20];

const toplam = koordinatlar.reduce( (tp, koordinat) => {
  console.log(`iterasyon ${tp} ${koordinat}`);
  return tp + koordinat;
}, 0);
console.log(toplam);
```

ÖRNEK:

Koordinatlar dizisindeki değerlerin toplamını, ara değerleri de göstererek konsola bastıran uygulamayı **reduce()** ile yazınız.



Iteration – reduce

ÖRNEK:

Koordinatlar dizisindeki değerlerin ortalamasını hesaplayarak konsola bastıran uygulamayı **reduce()** ile yazınız.

```
const koordinatlar = [-100, 150, -32, 43, -20];

let ortalama = koordinatlar.reduce((toplam, koordinat) => toplam + koordinat );
ortalama /= koordinatlar.length;

console.log("Koordinatların Ortalaması:" + ortalama);
```



Iteration – reduce

ÖRNEK:

Bir Firma, **3000 TL** den **az** olan maaşlara **%10** zam yapmak istiyor ve zam yapılan bu kişilere **toplam** kaç TL ödeneceğini bilmek istiyor. İlgili programı yazınız.

```
const maaslar = [3000, 2891, 3500, 4200, 7000, 2500];

const zamliToplam = maaslar
  .filter((maas) => maas < 3000)
  .map((maas) => maas * 1.1)
  .reduce((toplam, maas) => toplam + maas);

console.log(zamliToplam.toFixed(2));
```



Reduce

PRACTISE

Ülke listesinin altına ülke toplam yüzölçümlerini reduce kullanarak yazdırınız.



Iteration – every

Mevcut bir dizi içinde tüm elemanların istenilen kriteri sağlayıp sağlamadığını döndürür. Sağlıyorsa true, değilse false döner.

```
dizi.every( (item) => item > 33 );
```



Iteration – some

Mevcut bir dizi içinde en az bir elemanın istenilen kriteri sağlayıp sağlamadığını döndürür. Sağlıyorsa true, değilse false döner.

```
dizi.some( (item) => item > 33 );
```



JavaScript

DOM
(Document Object Model)

- › DOM nedir?
- › DOM yapısı
- › DOM erişim
- › DOM güncelleme



DOM Nedir?

DOM (Document Object Model), dokümanlarının **stillerinin, yapısının, içeriğinin** erişilmesine ve güncellenmesine olanak sağlayan bir nesne modelidir.

Objects

HTML
elemanları

Properties

HTML
elemanlarının
özellikleri
(attribute)

Methods

HTML
elemanlarına
erişmeyi
sağlayan
methodlar

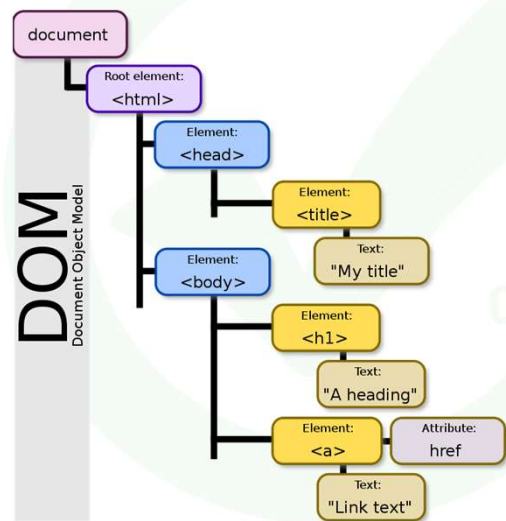
Events

HTML
elemanları
üzerinde
gerçekleşen
olaylar



DOM Yapısı

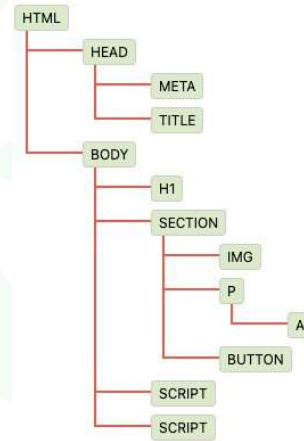
- ◉ Bir web sayfası yüklendiğinde tarayıcı, ilgili HTML sayfasının **DOM'unu** oluşturur.
- ◉ DOM ağaç yapısı ise HTML dosyası içerisinde bulunan **tüm nesnelerin** hiyerarşik bir biçimde gösterimidir.





DOM Yapısı

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>DOM</title>
</head>
<body>
  <h1>DOM Temelleri</h1>
  <section>
    
    <p>Javascript öğrenmek için link
      <a href="http://www.javascript.org">Javascript</a>
    </p>
    <button>Tıkla</button>
  </section>
  <script src="main.js"></script>
</body>
</html>
```



DOM'a Erişim

Method	Açıklama
getElementById	Bir elemanı id sinden bulur
getElementsByTagName	Bir elemanı Tag (Etiket) adına göre bulur
getElementsByClassName	Bir elemanı Class (Sınıf) adına göre bulur
querySelector	Bir elemanı id, class, özellik, tür ve değere göre seçmek için kullanılır. Eşleşen ilk elemanı seçer.
querySelectorAll	Bir elemanı id, class, özellik, tür ve değere göre seçmek için kullanılır. Eşleşen elemanların listesini döndürür.



DOM erişim

PRACTISE

Adı	Puanı
Ali	70
Veli	30
Ayşe	5
Fatma	90

Puanı belli bir değerden aşağıda olan kişilerin rengini kırmızı yapan programı yapınız.



DOM'a Erişim

Method	Açıklama
closest	En yakın elemanı seçer, yukarı doğru çalışır
children	Bir elementin çocuklarını seçer
firstElementChild, lastElementChild	Bir elementin ilk çocuğunu veya son çocuğunu seçer.
parentElement	Bir elemanın parent ını seçmek için kullanılır.
nextElementSibling, previousElementSibling	Bir elemanın önceki veya sonraki kardeşlerini (aynı seviye) seçmek için kullanılır.



DOM erişim

PRACTISE

Adı		
Ali		
Veli	30	Sil
Ayşe	5	Sil
Fatma	90	Sil

This page says
Ayşe isimli kaydı silmek istediğinizden emin misiniz.

OK Cancel

Her satırın sonundaki Sil butonlarına basıldığında satır başındaki ismi alarak uyarı veren programı yazınız.



DOM güncelleme

Method	Açıklama
innerText	Bir elemanın içindeki metni getirir veya içine metin yerleştirir.
innerHTML	Bir elemanın içindeki html i getirir veya içine HTML yerleştirir.
textContent	innerText ile aynıdır. Tek farklı stil özelliklerinden dolayı gizlenmiş olan text leri de getirir



DOM Styles

Method	Açıklama
classList	Bir elemanın class listesine erişim sağlar.
add("class")	classList e bir class ekler
remove("class")	classList teki bir class ı siler
toggle("class")	classList te yoksa class ı ekler, varsa class ı siler
contains("class")	classList te bir class ın olup olmadığını kontrol eder.
replace("class1", "class2")	classList te class1 i siler, yerine class2 yi koyar



DOM Styles

Bir tablonun satırlarına bir kere tıklandığında seçili hale getiren tekrar aynı satıra tıkladığında seçili olmaktan çıkaran programı yapınız.

PRACTISE

İsim	Puan	#
Ayşe	80	<button>Sil</button>
Ali	50	<button>Sil</button>
Veli	30	<button>Sil</button>
Fatma	20	<button>Sil</button>
Aslı	55	<button>Sil</button>



DOM Attributes

Method	Açıklama
<code>getAttribute("attribute")</code>	Bir elemanın içine metin yerleştirir.
<code>setAttribute("attribute", "value")</code>	Bir elemana attribute ekler/değiştirir
<code>removeAttribute("attribute")</code>	Bir elemanın attribute unu siler
<code>attributes</code>	Bir elemanın tüm attribute larını getirir.



DOM Attributes

PRACTISE

Güncelle

Default olarak disabled olan bir buton, textbox içine bir şey yazılınca aktif, silinince pasif olacak.



DOM güncelleme

Method	Açıklama
createElement("h1")	Yeni bir element oluşturur. Document objesinin bir methodudur.
appendChild(element)	Bir elemana başka bir eleman ekler
append(element, "metin")	Bir elemanın içinde en sona başka bir elemanı veya text i ekler. Tek seferde birden fazla ekleme yapılabilir.
prepend(element, "metin")	Bir elemanın içinde en başa başka bir elemanı veya text i ekler. Tek seferde birden fazla ekleme yapılabilir.
remove()	Bir elemanı siler




DOM

PRACTISE

Adı	Puanı	
Fatma	90	Sil
Ali	70	Sil
Veli	30	Sil
Ayşe	5	Sil

Tablodaki satırların sonundaki sil butonuna basıldığında satırı silen programı yapınız.



DOM

PRACTISE

Adı	Puanı	
Fatma	90	<button>Sil</button>
Ali	70	<button>Sil</button>
Veli	30	<button>Sil</button>
Ayşe	5	<button>Sil</button>

Puana tıklandığında puanı büyükten küçüğe doğru satırları sıralayan programı yapınız.



JavaScript

Advanced JavaScript

- › Error handling
- › Debugging
- › Timeout , Interval
- › Web storage
- › Rest & spread operatör
- › Destructuring
- › Callbacks
- › Fetch
- › Modules



Error handling

Hata olması muhtemel kodlarda, hataların programın akışını engellememesini sağlamak ve kullanıcıya daha kontrollü ve anlaşılır uyarılar vermek için yapılan işlemdir.

Genellikle kullanıcıdan bilgi alırken veya bir API dan bilgi çekerken kullanılırlar.

Try – Catch - Finaly



Error handling

```
try{
```

```
}
```

Hata olması beklenen kodlar try bloğu içine yazılır.

```
catch(ex){
```

```
}
```

Try bloğu içinde bir hata olduğunda program akışı catch bloğuna geçer ve hata ile ilgili bilgi ex object içinde bulunur.

```
finally {
```

```
}
```

Hata olsa da olmasa da çalıştırılan bir bloktur. Kullanılması zorunlu değildir.



Error handling

PRACTISE

Login

Username:

Password:

Login Register

Bir login formu oluřturup, form handling yapınız



Debugging



Programdaki hataların ayıklanması iřlemine debugging denir. Debugging iin chrome developer tools kullanılabileceęi gibi Vscodde gibi programlardaki debugger da kullanılabilir.



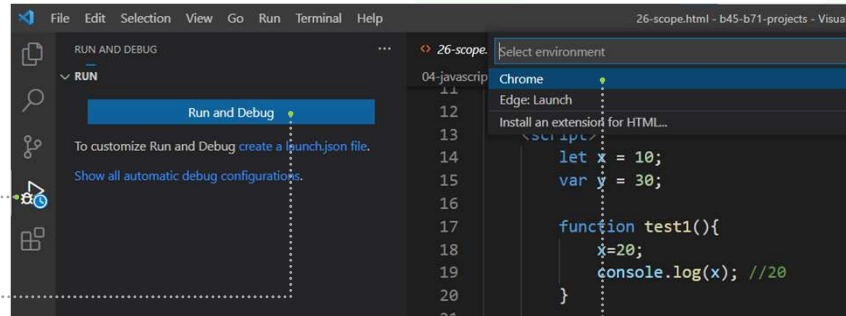
Debugging

1

Debugging bölümü açılır

2

Run and Debug butonuna basılır

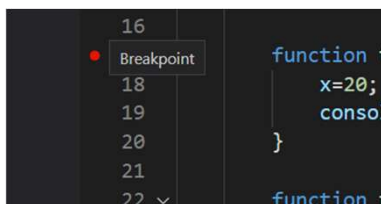


3

Hangi ortamda çalıştırılacağı seçilir



Debugging



- Program işleyişini belli bir noktada kesmek için breakpoint ler kullanılır.
- Program akışı o noktaya geldiğinde kesilecek ve programın adım adım çalıştırılabilmesi için debugging başlayacaktır.
- Breakpoint eklemek için satır numaralarının sol tarafındaki boşluğa tıklanır. Kırmızı noktalar breakpoint leri temsil eder.



Debugging

Değişken takip alanı

Continue Step Over Step Into Step Out Restart Stop

```

RUN AND DEBUG
No Configurations
26-scope.html
04-javascript > 26-scope.html > html > body
12
13
14   <script>
15     let x = 10;
16     var y = 30;
17
18     function test1(){
19       x=20;
20       console.log(x); //20
  
```

Variables

- Local: test1
 - this: Window
- Script
 - x: 20
- Global

WATCH



Timeout

Gecikmeli olarak çalışması istenilen kodlar, timeout oluşturularak ve belli bir gecikme zamanı belirlenerek çalıştırılabilir.

```

setTimeout( ()=> {
  // Gecikmeli
  // çalışacak
  // kodlar
  // burada
  // olacak
}, 500);
  
```

setTimeout methodu
ile bir zamanlayıcı
oluşturulur.

Gecikme zamanı ms
cinsinden tanımlanır

clearTimeout()
metodu ile
timer
durdurulabilir.



Timeout

PRACTISE

OPEN

Sayfanın üst tarafında normalde gözükmeyen ancak kırmızı alanın üstüne gelince aşağı doğru açılan menü yapınız. Fare alanın üzerinden ayrılınca gecikmeli kapansın.



Interval

Belli periyotlarla çalışması istenilen kodlar, interval oluşturularak ve belli bir zaman belirlenerek çalıştırılabilir.

```
setInterval( ()=> {  
  // Belli aralıklarla  
  // çalışacak  
  // kodlar  
  // burada  
  // olacak  
}, 500);
```

setInterval methodu
ile bir zamanlayıcı
oluşturulur.

Period ms cinsinden
tanımlanır

clearInterval()
methodu ile
timer
durdurulabilir.



Timeout

PRACTISE

11:17

Çalışan digital saat uygulaması yapınız



Web Storage API

Kullanıcı tarafında data saklayabilmek için HTML5 ile birlikte gelen Storage API kullanılır.

localStorage

Veri, kullanıcı veya app silene kadar saklanır.

sessionStorage

Veri, oturum bazlı saklanır. Tarayıcı kapatıldığında veri silinir.



Web Storage API

localStorage ve sessionStorage de data manipölasyonu için kullanılan methodlar

setItem

Veri saklamak için

getItem

Veriyi almak için

removeItem

Veriyi silmek için

```
localStorage.setItem("counter", 1);
```



Web Storage API

Sayfa her ziyaret edildiğinde, ziyaretçiye siteye kaçınıcı kez girdiğini söyleyen programı yazınız.

PRACTISE



Rest & Spread Operator

ES
6

Rest ve Spread operatörleri 3 nokta ile ifade edilirler ancak birbirinden farklıdır.

REST

Farklı değişkenlerden gelen verileri bir dizi içinde toplar

SPREAD

Dizi veya object içindeki verileri ayrı değişkenlere açar.



Rest Operator

```
function topla(...sayilar){  
  return sayilar.reduce( (toplam, x)=> toplam + x);  
}  
  
console.log(topla(3,5,6));  
console.log(topla(13,5,16,3,56,76,1));
```

Gelen tüm parametreleri sayılar isimli dizi içinde alacaktır.



Rest Operator

```
function ortalama(ad, soyad, ...notlar){  
  let ort = notlar.reduce( (toplam, x)=> toplam + x);  
  return `${ad} ${soyad} ortalaması: ${ort/notlar.length}`  
}  
  
console.log(ortalama("Ali", "Gel", 3, 5, 6));  
console.log(topla(("Ayşe", "Koş", 13, 5, 16, 3, 56, 76, 1)));
```

İlk iki parametre haricinde geri kalan tüm parametreleri bir dizi içine alacaktır



Spread Operator

```
function topla(s1, s2, s3){  
  return s1 + s2 + s3;  
}  
  
const sayilar = [12, 56, 17];  
console.log(topla(...sayilar));
```

Sayılar dizisini genişletip her bir elemanını ayrı bir değişken haline getirdi.



Spread Operator

```
let numberStore = [0, 1, 2];  
let newNumber = 12;  
numberStore = [...numberStore, newNumber, 15];
```

numberStore dizisi başka bir dizi içinde açılarak ve yeni elemanlar eklenerek güncellendi.



Spread Operator

```
const student = {  
  firstName: "Ali",  
  lastName: "Gel",  
  no: "234",  
  phone: "23423423423"  
};  
  
let newStudent = {...student, firstName: "Veli"};  
console.log(student, newStudent);
```

student nesnesini genişletip, firstName özelliğinin değeri değiştirip yeni bir nesne oluşturuyor.



Destructuring

ES
6

Bir nesnenin değerlerinin değişkenlere açılması işlemine destructuring denir.

```
let { variable1, variable2 } = object;
```



Destructuring

```
const student = {  
  firstName: "Ali",  
  lastName: "Gel",  
  no: "234",  
  phone: "23423423423"  
};
```

```
const { firstName, lastName } = student;  
console.log(firstName);
```

Student nesnesi içindeki, "Ali" ve "Gel" değerlerini ayrı ayrı firstName ve lastName değişkenlerine aktarmış



Synchronous vs Asynchronous

Javascript senkron bir yapıda çalışır. Yani kodlar sırayla çalıştırılmaktadır. Ancak bazı durumlarda asenkron moda geçerek çalışma hızını artırır.

setTimeout

setInterval

fetch



Synchronous vs Asynchronous

```
function getData(){  
  setTimeout( ()=> {  
    console.log("ok2");  
  },500);  
}
```

```
console.log("ok1");  
getData();  
console.log("ok3");
```



ok1
ok3
ok2



Callbacks

Asenkron çalışan fonksiyonlardan veri döndürmek için callback fonksiyonlar kullanılır.

```
...  
...  
...  
getData( (data)=> {  
  
});
```

```
function getData(callback){  
  ...  
  callback("dönen data");  
}
```



Callbacks

```
function getData(callback){  
  setTimeout( ()=> {  
    callback("ok2");  
  },500);  
}  
  
console.log("ok1");  
getData( (msg)=> {  
  console.log(msg);  
});  
console.log("ok3");
```



ok1
ok3
ok2



Fetch

Harici bir kaynaktan data çekmek için kullanılan asenkron çalışan bir yapıdır. Bu harici kaynak bir API veya dosya olabilir.

```
fetch( kaynak_url)
.then(resp=>resp.json())
.then(data=>console.log(data))
.catch(err=>console.log(err));
```



Fetch

Bir API ye bağlanarak data çekiniz

PRACTISE



Modules

Modüller, JS kodlarını bölerek farklı farklı dosyalardan çağırmak ve böylece büyük projelerde kod karmaşasının önüne geçmek için kullanılan bir yöntemdir.

export

Dışarı aktarma

import

İçeri alma

Dışarı aktarılan kodlar içeri alınabilir.



Modules

app.js

```
import { showAlert, showConfirm }  
from "../message.js";  
  
showAlert();
```

message.js

```
export const showAlert = () => {  
}  
  
export const showConfirm = () => {  
}
```



Modules

app.js

```
import {showAlert, showConfirm}
from "./message.js";

showAlert();
```

message.js

```
const showAlert = () => {
}

const showConfirm = () => {
}

export {showAlert, showConfirm};
```



Modules

app.js

```
import showAlert, {showConfirm}
from "./message.js";

showAlert();
```

message.js

```
const showAlert = () => {
}

const showConfirm = () => {
}

export default showAlert;
export {showConfirm};
```