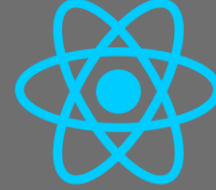


FRONT - END



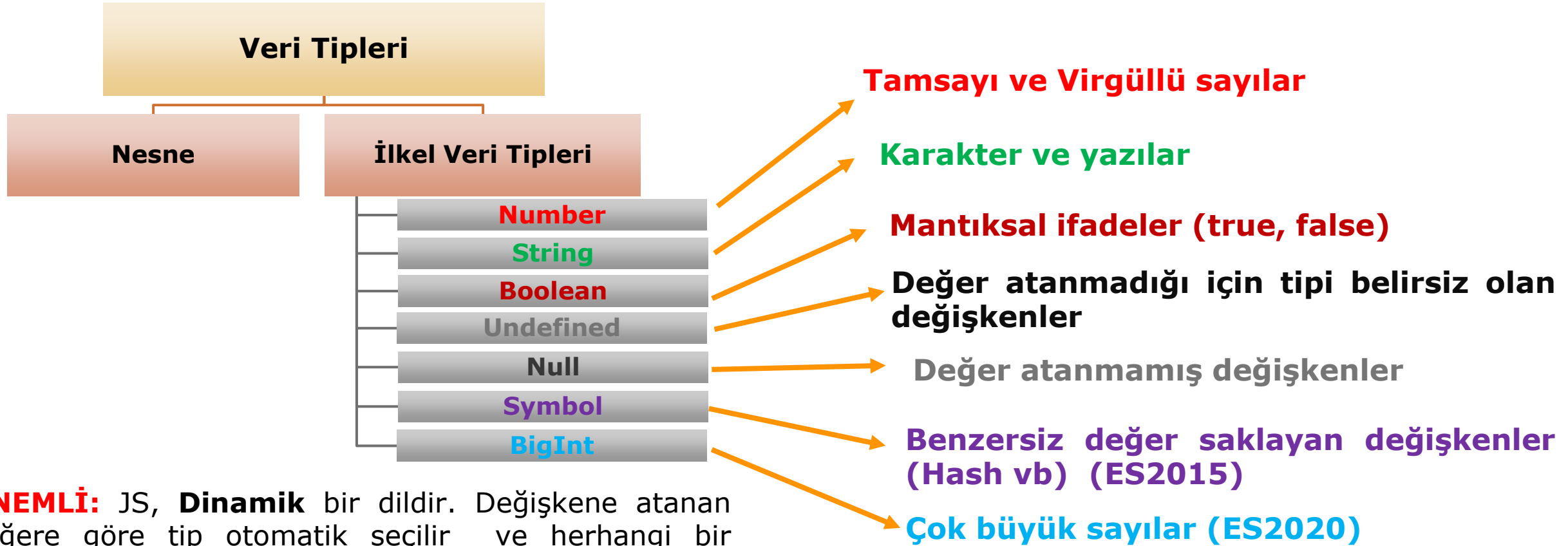
JAVASCRIPT



DEĞİŞKENLER, OPERATÖRLER

JS VERİ TIPLERİ

- Javascript'te değişkenleri **2 ana** kategoride inceleyebiliriz.



ÖNEMLİ: JS, **Dinamik** bir dildir. Değişkene atanan değere göre tip otomatik seçilir ve herhangi bir zamanda değiştirilebilir.

DEĞİŞKEN TANIMLAMA

- Javascript'te değişken tanımlamak için **3 adet** anahtar kelime (keyword) bulunmaktadır.
 - **var, const ve let**
 - **let** ve **const**, **ES6** ile eklenmiştir (Modern Javascript).
 - Modern JS öncesinde, sadece **var** kullanılıyordu.
- **TANIMLAMA**

AnahtarKelime değişkenİsmi = başlangıçDeğeri;






const pi = 3.14;

var sayac = 1;

let yaş = 33;

NOT: Satırı sonu için ; zorunlu değildir. Genelde programcılar tercih etmektedir. Değişken ismi verirken diğer dillerde olduğu gibi bazı kurallar vardır.

DEĞİŞKEN TANIMLAMA KURALLARI

- Değişkenin ilk karakteri harf yada alt çizgi olmalıdır. Sayı ile başlayamaz.
 - const** sayı1 = 5;  **let** 1sayı = 3; 
- Değişken adının geri kalan kısmı harf, rakam ve alt çizgi içerebilir. Ama boşluk, sembol ve özel işaretler içeremez.
 - let versionNo = 1**  **var** öğrenci no = 1001; 
- JS küçük-büyük harf duyarlı (case-sensitive) bir dildir.
 - Sayı  sayı

Değişken tanımlarken Javascript'in **ayrılmış kelimelerini** kullanamayız.

- let** for = 5;   **let** do = 10;

CONST

- Sadece başlangıç ataması ile değer atanabilen sonrasında değeri değiştirilemeyen değişken tanımlamasıdır.
- Javadaki **FINAL** tanımlamasına benzer (Read-Only).
- Sadece tanımlandığı blok içerisinde geçerlidir. Diğer yerlerden erişilemez (**Block-Scoped**).
- **Avantajı** : Tanımlama dışında değeri değiştirilemediği için güvenlidir. Hata ile değer atılamaz.
- **Dezavantajı**: Değeri tekrar-tekrar değişecek verileri saklamak için kullanılamaz.
- Eğer mümkünse yani bir değişkenin değeri bir daha değişmeyecekse güvenlik için **const** kullanmakta fayda var.

CONST

```
const isim = "Clarysway";  
console.log(isim);  
console.log(typeof isim1);
```

→ **Clarysway** yazdırır

→ **string** yazdırır

```
const pi = 3.14;  
console.log(pi);  
console.log(typeof pi);
```

→ **3.14** yazdırır

→ **number** yazdırır

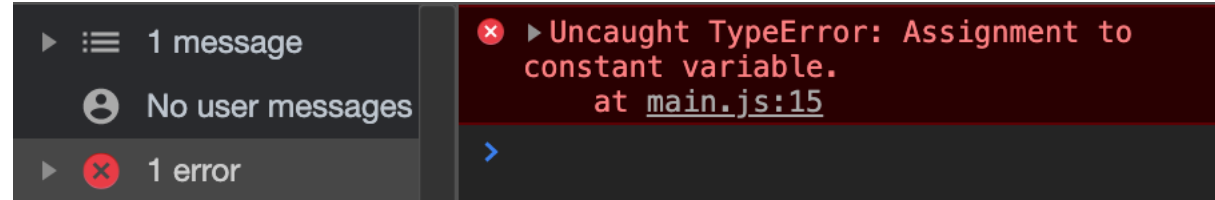
```
const dogruMu = true;  
console.log(dogruMu);  
console.log(typeof dogruMu);
```

→ **true** yazdırır

→ **boolean** yazdırır

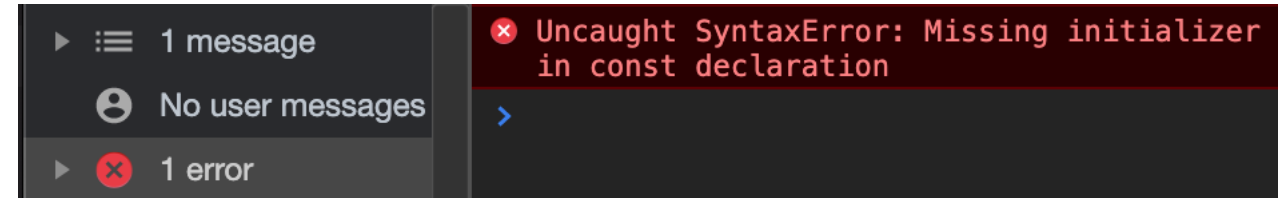
CONST

```
const sayı = 5;  
sayı = 7;
```



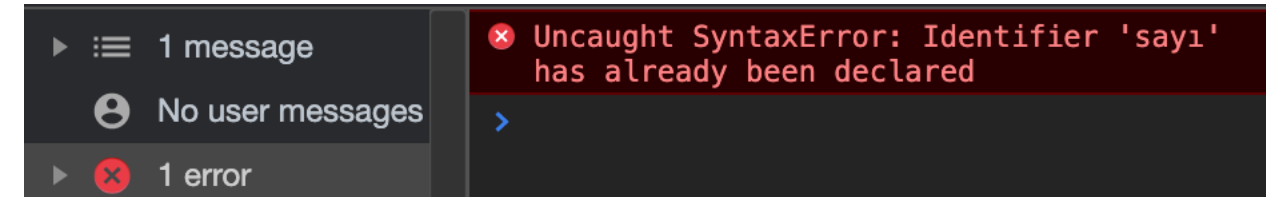
HATA: **const** değişkene sonradan değer atanamaz.

```
const ad;
```



HATA: **const** değişkene başlangıç değeri atanmalıdır.

```
const sayı = 2;
```



HATA: **const** deyimiyle, aynı-scope'da aynı isimle değişken tanımlanamaz.

LET

- **const'dan** farkı, istenildiği zaman değerinin değiştirilebilmesidir.
- **const** gibi tanımlandığı blok içerisinde geçerlidir. Başka yerlerden erişilemez (**Block-Scoped**).
- Bir değişken aynı isimle tekrar tanımlanamaz.
- **const** kullanamadığımız durumlarda (değişkenin değeri değişecekse) **let** kullanmalıyız.

LET

```
let dil = "Java";  
dil = "Javascript";  
console.log(dil);  
console.log(typeof dil);
```

→ **Javascript** yazdırır
→ **string** yazdırır

```
dil = 1;  
console.log(dil);  
console.log(typeof dil);
```

→ **1** yazdırır
→ **number** yazdırır

```
dil = true;  
console.log(dil);  
console.log(typeof dil);
```

→ **true** yazdırır
→ **boolean** yazdırır

```
dil = null;  
console.log(dil);  
console.log(typeof dil);
```

→ **null** yazdırır
→ **object** yazdırır

LET

```
let sayaç;  
console.log(sayaç);  
console.log(typeof sayaç);
```

→ **undefined** yazdırır

→ **undefined** yazdırır

```
sayaç = true  
console.log(typeof sayaç);
```

→ **boolean** yazdırır

```
kalanBorç = 15.5;  
console.log(kalanBorç);  
console.log(typeof kalanBorç);
```

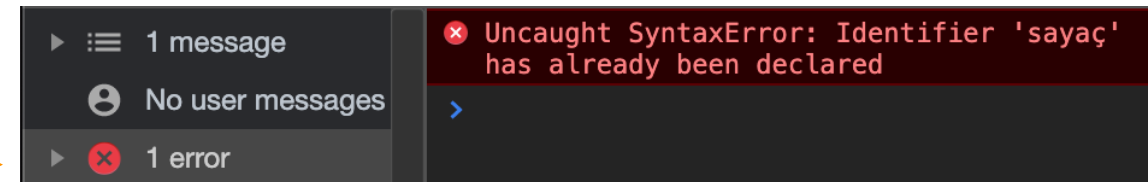
→ **15.5** yazdırır

→ **number** yazdırır

ÖNEMLİ:

let kullanılsa bile JS yorumlayıcısı yeni değişkeni algılayıp tanımlar.

```
let sayaç = 2
```



Hata: **let** ve **const** deyimiyle aynı scope'da tekrardan aynı isimle değişken tanımlanamaz.

VAR

- **var** deyimi ile tanımlanan değişkene, **let** de olduğu gibi tekrardan değer atamak mümkündür.
- **var**'ın **let**'ten farkı, **global-scope** veya **fonksiyon-scope** olmasıdır.
 - Eğer bir fonksiyon içerisinde tanımlandı ise sadece o fonksiyonda geçerlidir. (**fonksiyon-scope**).
 - Fonksiyon dışında tanımlandı ise her yerde geçerlidir (**global scope**).
- **var** ile aynı isimle tekrar değişken tanımlamak mümkündür.
- Günümüzde programcılar global değişken gerekmedikçe **var** ile değişken tanımlamamayı seçmektedir.

VAR

```
var fiyat;  
fiyat = 23;  
console.log(fiyat);
```

→ **23** yazdırır

```
fiyat = 19.99;  
console.log("FİYAT:" + fiyat);
```

→ **FİYAT: 19.99** yazdırır

```
var fiyat = "ücretsiz";  
console.log("FİYAT:" + fiyat);
```

→ **FİYAT: ücretsiz** yazdırır

ÖNEMLİ: **var** deyimi ile aynı isimle tekrar değişken tanımlamak mümkündür.

NOT: **var** ile **let** arasındaki en büyük fark **scope** yani geçerli oldukları bölge farkıdır.

- **let** ile tanımlanan değişken sadece tanımlandığı blok içinde (döngü, fonksiyon v.b) geçerlidir.
- **var** ile tanımlanan değişken ise ya **global** bir değişkendir yada fonksiyon içerisinde geçerlidir.

OPERATÖRLER

- Javascript operatörlerini 5 kategoride inceleyebiliriz.
 - Aritmetik Operatörler
 - Karşılaştırma Operatörleri
 - Mantıksal Operatörleri
 - Atama Operatörleri
 - Koşul (Ternary) Operatörü

ARİTMETİK OPERATÖRLER

| Operatör | Açıklaması |
|----------|---|
| + | Toplama işlemi ve String'lerde birleştirme işlemi gerçekleştirir. |
| - | Çıkarma işlemi gerçekleştirir |
| * | Çarpma işlemi gerçekleştirir |
| % | Mod alma işlemi gerçekleştirir. |
| ++ | Bir arttırma işlemi gerçekleştirir. |
| -- | Bir azaltma işlemi gerçekleştirir. |
| ** | Üs alma işlemi gerçekleştirir |

ARİTMETİK 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 = 'Mustafa';  
const soyAd = 'Çalışkan';  
console.log(ad + soyAd);  
console.log(ad + ' ' + soyAd);
```

ÖNEMLİ

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

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

ÇIKTI

55

BACKTICK (`)

- **String'leri** daha dinamik bir şekilde birleştirmek için **ES6** ile yeni gelen **String Şablonları** (Template literals) kullanabiliriz.
- Tek tırnak (') ve çift tırnağa (") ek olarak **backtick** (`) ile de **string** tanımlamak mümkündür.
 - **`string ifade`**
 - **`string \${değişken} string`**



NOT

Klavyelere göre yeri değişebiliyor.

```
console.log(`HARCAMA:${toplamHarcama} TL`);  
  
const sonuç = `HARCAMA:${toplamHarcama} TL`;  
console.log(sonuç);
```

ÇIKTI

HARCAMA: 72 TL

NOT: Bir çok kolaylığa imkan sağlayan bu konuyu, ileride daha ayrıntılı olarak ele alacağız.

ARİTMETİK OPERATÖRLER (-)

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

ÇIKTI
YAŞ: 41

ÖNEMLİ

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

NaN = Not a Number
(Sayı değil)

ARİTMETİK OPERATÖRLER (* VE **)

```
const pi = 3.14;  
const r = 3;  
const alan = pi*r**2;  
const çevre = 2*pi*r  
console.log(çevre, alan);  
console.log("ÇEVRE:" + çevre, "ALAN:" + alan);
```

NOT: ** Üs alma işlemi gerçekleştirir.

ARİTMETİK 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++;
```



HATA: **const** değişkenin değeri arttırılamaz.

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



ÇIKTI
Birler Basamağı: 3

KARŞILAŞTIRMA OPERATÖRLERİ

| Operatör | Açıklaması |
|--------------|---|
| == | İki değişkenin veri tipine bakmaksızın eşitliğini kontrol eder. Eşitse true aksi takdirde false döndürür. |
| === | Veri tipi de dahil olmak üzere eşitliğini kontrol eder. Eşitse true aksi takdirde false döndürür. |
| != | İki değişkenin eşit olmamasını kontrol eder. Eşit değilse true aksi takdirde false döndürür. |
| > | Soldaki değişkenin değeri sağdakinden büyükse true aksi takdirde false döndürür. |
| < | Soldaki değişkenin değeri sağdakinden küçükse true aksi takdirde false döndürür. |
| >= | Soldaki değişkenin değeri sağdakine eşit veya büyükse true aksi takdirde false döndürür. |
| <= | Soldaki değişkenin değeri sağdakine eşit veya küçükse true aksi takdirde false döndürür. |

KARŞILAŞTIRMA OPERATÖRLERİ

```
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 veri tipini de kontrol eder.

ÖNEMLİ:

JS, operatörlere bakarak gerektiğinde **string** formatındaki sayıyı **number** formatına çevirerek işlemi gerçekleştiriyor.

ÖNEMLİ:

Büyük eşit ve küçük eşit işlemlerinde veri tipi kontrolü yapılamıyor.

MANTIKSAL OPERATÖRLER

| Operatör | Açıklaması |
|-------------------|---|
| && | MANTIKSAL VE işlemi gerçekleştirir. Kontrol ettiği değişkenlerin tamamı TRUE ise TRUE değer döndürür. Aksi takdirde FALSE değer döndürür. |
| | MANTIKSAL VEYA işlemi gerçekleştirir. Kontrol ettiği değişkenlerin sadece bir tanesi bile TRUE ise TRUE değer döndürür. Ancak tamamı FALSE ise FALSE değer döndürür. |
| ! | MANTIKSAL DEĞİL işlemi gerçekleştirir. Yani, kontrol ettiği değişkenin değerinin tersini döndürür. Değişken TRUE ise FALSE , FALSE ise TRUE değer döndürür. |

ÖNEMLİ: **&** ve **|** operatörleri **Bit-temelli VE ,VEYA** işlemi gerçekleştirir.

MANTIKSAL OPERATÖRLERİ

```
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

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

ÖNEMLİ:

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

MANTIKSAL OPERATÖRLERİ

```
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 değişkenin değeri döndürülür.
- **VE** işleminde ise en sona kadar kontrol edilmesi gerekir. Dolayısıyla, hepsi doğru ise en sondaki değişkenin değeri döndürülür.

ATAMA OPERATÖRLERİ

| Operatör | Örnek | Açıklaması |
|----------|-------------------|---|
| = | x = y | Soldaki değişkenin değerini sağdakine 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. |

TİP DÖNÜŞÜMLERİ

```
const para = "1000";  
console.log(para + 15);  
console.log(Number(para) + 15);
```

ÇIKTI
10015

ÇIKTI
1015

ÖNEMLİ

Number() fonksiyonu
tip çevrimi yapılabilir.

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

ÇIKTI
NaN

ÖNEMLİ

Number() fonksiyonu Harfleri
sayıya çeviremediği için **NaN**
(Sayı değil - Not a Number)
döndürüyor.

```
const sayi = 54;  
console.log(String(sayi),sayi);
```

ÇIKTI
54 54

ÖNEMLİ

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

TİP DÖNÜŞÜMLERİ

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

ÇIKTI
true

ÖNEMLİ

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

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

```
const sıfır = 0 , nal = null;  
const tanımsız = undefined;  
const boş = "" , sayıDeğil = NaN;  
  
console.log(Boolean(sıfır), Boolean(nal));  
console.log(Boolean(tanımsız), Boolean(baş));  
console.log(Boolean(sayıDeğil));
```

ÇIKTI

false false

false false

false

ÖDEV

- **const** ile **let** arasındaki farklar nelerdir?
- **let** ile **var** arasındaki farklar nelerdir?
- **==** ile **===** farkı nedir?
- 5 adet falsy değeri içeren veri tipleri nelerdir?