Karşılaştırmalar

Çoğu karşılaştırma operatörlerini matematik derslerinden biliyorsunuzdur:

- Büyüktür/küçüktür: a > b, a < b.
- Büyük Eşit/Küçük Eşit: a >= b, a <= b.
- Eşitlik kontrolü a == b (Dikkat ederseniz tek değil iki tane '=' işaretinden oluşuyor.Tek olanı a = b atama anlamına geliyor).
- Eşit değildir matematikte şu şekilde gösteriliyor ≠, JavaScript'te ise eşittir öncesine ünlem işareti olarak kullanabilirsiniz a != b.

Sonuç boolean olacaktır

Diğer operatörler gibi bunun sonucu da değer dönecektir. Dönen değer booleandır.

- true – "evet", "dogru" veya "gerçek" demek.
- false – "no", "yanlış" veya "yalan" demektir.

Örneğin:

```
alert( 2 > 1 );  // true (correct)

alert( 2 == 1 ); // false (wrong)

alert( 2 != 1 ); // true (correct)
```

Bu karşılaştırmanın sonucu da diğer değerler gibi elbette değişkene atanabilir.

```
let sonuc = 5 > 4; // karşılaştırma sonucu oluşacak sonucu değişkene atayıp

alert( sonuc ); // ekranda gösterdiğimizde "true" - doğru sonucunu göreceksiniz.
```

1. Karakter dizisi karşılaştırma

Hangi karakter dizisinin hangisinden büyük olduğunu bulmak için alfabe sırasına bakılır.

Her harf sıra ile kontrol edilir.

Örneğin:

```
alert( 'Z' > 'A' ); // doğru ( true )

alert( 'Kum' > 'Kan' ); // doğru ( true )

alert( 'Bee' > 'Be' ); // doğru ( true )
```

İki karakter dizisini karşılaştıran algoritma aslında basit bir algoritmadır. Basamakları şu şekildedir.

1. İki karakter dizisinin ilk karakterlerini karşılaştır
2. Eğer birincisi ikincisinden büyükse, birinci karakter dizisi büyüktür. İşlem bitti.
3. Eğer karakterler eşit ise ikinci karakteri de aynı şekilde kontrol et. Eğer birincisi büyükse true döner.
4. Karakter dizilerinin sonuna kadar test et.
5. Eğer sonuna kadar tüm karakterler aynıysa uzun olanı daha büyüktür.

Örneğin birinci örnekte 'Z' > 'A' dan büyüktür hemen true sonucu döner.

İkincisinde "Kum" ve "Kan" karakter karakter karşılaştırılıyor:

1. K eşittir K 'ya.

2. u, a'dan büyük. Burada bitirilir ve birincisi ikincisinden büyüktür sonucu doğru(true) döner. Tam olarak sözlükteki sıralama gibi değildir, unicode bir sıralama var

Algoritmada belirtilen karşılaştırma tam olarak sözlükteki veya telefon defterindeki gibi bir karşılaştırma vermez.

Örneğin karakterin büyük veya küçük harf olması önemlidir. "A" ile "a" eşit değildir. Peki hangisi büyük? Aslında "a" daha büyüktür. Neden ? Çünkü küçük harf karakterler büyüklere göre (Unicode) index olarak daha sonradır. Bu konuya Karakter Dizisi - Strings bölümünden daha ayrıntılı bir şekilde inceleyebilirsiniz.

2. Farklı tiplerin karşılaştırılması

Farklı tipler karşılaşılaştırıldığında sayılara döünüştürülürler.

Örneğin:

```
alert( '2' > 1 ); // doğru, karakter olan `2` sayıya çevrilerek 2 olmuş ve böyle karşılaştırılmıştır.

alert( '01' == 1 ); // doğru, karakter olan '01' sayıya çevrilerek 1 olmuştur.

Boolan değerler için true 1 olur ve false 0 olur.

Örneğin:

alert( true == 1 ); // true ( doğru )

alert( false == 0 ); // true ( doğru )
```

Komik sonuçlar

Bazen biri true diğeri false olan iki değer eşit olabilir.

Örneğin:

```
let a = 0;

alert( Boolean(a) ); // false

let b = "0";

alert( Boolean(b) ); // true

alert(a == b); // true!
```

JavaScript tarafından bakınca aslında yukarıdaki sonuçlar gayet doğaldır. Eşitlik kontrolü sayısal karşılaştırmalara göre yapılıyor. Fakat boolean çevirme başka kurallara göre yapılmaktadır. Bundan dolayı Boolean("0") doğru döndürmektedir.

3. Sıkı Eşitlik

Normal eşitlik kontorlü "=="'nün bir problemi vardır. 0 ile false'ı birbirinden ayıramamaktadır.

```
alert( 0 == false ); // true
```

Aynı şey boş karakterler:

```
alert( '' == false ); // true
```

Bunun nedeni farklı tiplerdeki verilerin karşılaştırılmaya çalışılmasıdır. Her iki taraf da sayısal değerlere çevrilir. Örneğin " 0 olur, aynen false'ın 0 olması gibi.

Peki false ile 0'ın birbirinden farklı olarak tanımlamak için ne yapılmalıdır?

Sıkı eşitlik kontrolü === eşitliğin iki tarafını değiştirmeden kontrol eder.

Diğer bir deyişle eğer a ve b iki farklı tip ise a === b doğrudan false (yanlış) döner.

Örneğin:

```
alert( 0 === false ); // yanlış, çünkü tipler farklı.
```

Ayrıca eşitsizliği belirtmek için !== operatörü de bulunmaktadır.

Sıkı eşitlik kontrolü biraz daha uzun yazıma sahip olsa da hataya yer bırakmaz.

4. null ile undefined(tanımlanmamış) eşitlik kontrolü.

Daha uç noktalara bakarsanız,

null ile undefined başka değerler ile karşılaştırıldığında aralarında sezgisel olmayan davranışlar görülür.

Sıkı eşitlik kontrolü için ===: Bu değerler farklıdır, çünkü her biri kendine has bir tiptir.

```
```js run
alert( null === undefined ); // false
```
```

Sıkı olmayan eşitlik kontrolüne göre ==: bunlar birbirlerine eşit fakat başka hiçbir değere eşit değildirler.

```
```js run
alert( null == undefined ); // true
```
```

Matematiksel karşılaştırmalar için < > <= >= null/undefined sayıya çevrilirler. null 0 olurken undefined NaN(not a number) olur.

Şimdi bu bildiklerinizle aşağıdaki şakalı örneklere bakabilirsiniz.

0 ile null'in garip sonucu

Null ile 0'ın karşılaştırılması aşağıdaki gibidir:

```
alert( null > 0 );  // (1) false
alert( null == 0 ); // (2) false
alert( null >= 0 ); // (3) true
```

Matematiksel olarak yukarıda gördüğünüz örnekler imkansız. En sondaki örneğe bakarsanız 0'a eşit veya büyüklük durumunu kontrol ediyor. Eğer en alttaki doğru ise üsttekilerden en az birisinin doğru olması zorunludur. Fakat ikisi de yanlış.

Bunun nedeni eşitlik kontrolü == ve karşılaştırma kontrollerinin > < >= <= farklı çalışmasından dolayıdır. Karşılaştırma iki tarafta bulunan değerleri önce sayıya çevirmeye çalışır. Bundan dolayı sonuncu örnekte null>=0 null 0 a dönüşür. En üst örnekte de null>0 bu şekilde çalışır. Bundan dolayı en üstte false, en altta ise true döner.

Diğer bir taraftan; eşitlik kontrolü ==, undefinedve nulliçin kurala göre bir değişikliğe uğramaz. Sadece birbirleri arasında (undefined ile null) eşitliğe sahiptirler. Diğer türlü hiçbir şeye eşit değildirler. Bundan dolayı null == 0 false olur.

5. Karşılaştırılamaz tanımsız (undefined)

undefined hiçbir zaman karşılaştırma içerisine girmemelidir.

```
alert( undefined > 0 ); // false (1)

alert( undefined < 0 ); // false (2)

alert( undefined == 0 ); // false (3)
```

Neden hep false çıktı?

Bu sonuçları şunlardan dolayı aldık

- (1.) ve (2.) örneklerde false döndü çünkü undefined NaN oldu. Nan özel bir sayısal değişkendir ve hangi sayı ile karşılaştırılırsa karşılaştırılsın, sonuç false çıkar.
- (3.) maddedeki eşitlik kontrolü ise undefined'ın sadece null ile eşit olabilmesinden dolayıdır. null haricinde hiçbir değere eşit değildir.

Problemlerden Kaçınma

Neden peki bu örnekleri yaptık? Bu şeyleri her zaman hatırlamamıza gerek var mı? Aslında haklısınız bu gibi özelliklere zamanla daha iyi alışabilirsiniz. Fakat bu problemlerden kaçınmanın bir yolu var.

undefined/null eşitlik kontrollerinde sıkı eşitlik kontrolü === haricinde yaptığınız kontrollere dikkat etmeniz lazım.

>= > < <= gibi karşılaştırmaları null/undefined değeri alabilecek değişkenler ile yapmayın, yaparsanız bile kesinlikle çok dikkatli olun. Eğer bir değişken null/undefined gibi değerler alabiliyorsa bunları ayrıca kontrol etmeniz gerekli.

Özet

- Karşılaştırma operatörleri mantıksal değerler döndürür. (true/false) gibi
- Karakter dizileri harf harf alfabe sırasına göre kontrol edilir.
- Karşılaştırmalarda eğer farklı tipler kullanılıyorsa bu işlem yapılmadan sayıya çevirilir. (Eğer sıkı eşittir kullanıyorsanız çevirilmez)
- null ve undefined eşittir. Bu değerler başka hiçbir değere eşit değildirler.
- Değeri null/undefined olabilen bir değişken ile > veya < karşılaştırması yaparken dikkat edin. Ayrı bir null/undefined kontrolü yapmakta fayda var.

Görevler

Karşılaştırma

önem: 1

Aşağıdaki ifadelerin sonuçlarını yazınız?

```
5 > 4
```

```
"apple" > "pineapple"
```

```
"2" > "12"
```

```
undefined == null
```

```
undefined === null
```

```
null == "\n0\n"
```

```
null === +"\n0\n"
```