Veri Tipleri

Bir JavaScript değişkeni her türlü veriyi tutabilir. Önce karakter dizisi (string) atansa da sonra sayısal değer alabilir:

```
// Hata yok

let mesaj = "merhaba";

mesaj = 123456;
```

Bu şekilde olaylara izin veren tipteki dillere "dinamik tip" dil denir. Veri yapıları olsa bile değişkenler bu yapılara bağlı değildir.

JavaScript dilinde sekiz farklı veri tipi bulunmaktadır. Şimdilik bu tiplerden bahsedeceğiz gelecek bölümlerde ise daha derinlemesine bu tipleri inceleyeceğiz.

1. Number – Sayı

```
let s = 123;

s = 12.345;
```

sayı hem integer hem de floating point sayıları için kullanılır. Sayılar *, /, + veya - işlemlerine girebilirler. Normal sayıların haricinde "özel sayısal değerler" de sayı olarak tanımlanabilir. Bunlar : Infinity, -Infinity ve NaN gibi değerlerdir.

- Infinity matematiksel Sonsuzluğu ∞ ifade eder. Diğer tüm sayılardan büyük olan özel bir sayıdır.

0'a bölünmede sonuç sonsuzu verir:

```
alert( 1 / 0 ); // Infinity
```

veya doğrudan ekranda gösterebilirsiniz:

```
alert( Infinity ); // Infinity
```

- NaN hesaplamalarda bir hata olduğunu gösterir. Hatalı veya tanımsız matematiksel hesapları gösterir, örneğin:

```
alert( "Sayı Değil ( Not a Number) " / 2 ); // NaN, böyle bir bölme işlemi yapılamaz.
```

NaN yapışkandır. NaN üzerinde yapılacak herhangi bir işlem yeniden NaN çıktısı verecektir:

```
alert( "not a number" / 2 + 5 ); // NaN
```

Öyleyse matematiksel işlemlerin herhangi bir yerinde NaN alınıyorsa bu hesabın tamamını etkiler.

Matematiksel hesapların güvenliği

JavaScript üzerinden matematik hesapları yapmak güvenlidir. Her işlemi yapabilirsiniz. 0'a bölme normal harf dizesini bir sayıya bölmeye çalışma vs.

Kodunuzun tamamı hiç durmadan çalışacaktır. En kötü ihtimalle NaN sonucunu alınır.

Özel sayısal değerler "number" tipine aittir. Tabii ki sayı bizim bildiğimiz tipte sayı değillerdir. Sayılar bölümünde sayısal değerler ile çalışmayı daha derinlemesine göreceksiniz.

2. BigInt – Büyük Sayı

JavaScript'te "number" türü, şundan büyük tamsayı değerlerini temsil edemez (2^{53}-1) (bu 9007199254740991), veya daha az -(2^{53}-1) negatifler için. Dahili temsillerinden kaynaklanan teknik bir sınırlamadır.

Çoğu amaç için bu oldukça yeterlidir, ancak bazen gerçekten büyük sayılara ihtiyacımız olabilir, kriptografi veya mikrosaniye hassasiyetli zaman damgaları için.

Son zamanlarda, isteğe bağlı uzunluktaki tam sayıları temsil etmek için dile BigInt türü eklendi.

Bir tamsayının sonuna n eklenerek BigInt değeri oluşturulur:

```
// Sondaki "n" bu değerin bir BigInt olduğu anlamına gelir

const bigInt = 1234567890123456789012345678901234567890n;
```

BigInt sayılarına nadiren ihtiyaç duyulduğundan, onları burada ele almıyoruz, ancak onlara ayrı bir bölüm "bigint" adresindeki makale bulunamadı ayırdık. Bu kadar büyük sayılara ihtiyacınız olduğunda okuyun.

Compatibility issues

Şu anda, BigInt Firefox/Chrome/Edge/Safari'de destekleniyor, ancak IE'de desteklenmiyor.

Bir tarayıcının hangi sürümlerinin desteklendiğini öğrenmek için * MDN * BigInt uyumluluk tablosunu kontrol edebilirsiniz.

String – Karakter Dizisi

JavaScript'te karakter dizileri çift tırnak içerisine alınmalıdır.

```
let str = "Merhaba";

let str2 = 'Tek tırnak da çalışır';

let phrase = `değer gömülebilir ${str}`;
```

JavaScript'te 3 çeşit tırnak içine alma yöntemi vardır.

1. Çift tırnak: "Hello".
2. Tek tırnak: 'Hello'.
3. Ters tırnak: `Hello`.

Çift tırnak ile tek tırnak "basit" tırnaklardır. Aralarında bir farklılık yoktur.

Ters tırnak ise "genişletilmiş fonksiyonlu" tırnaktır. Bunu kullanarak karakter dizisi içerisine ${...} gibi başka bir dizi yerleştirebiliriz. Örneğin:

```
let isim = "Ahmet";

// değişken gömme

alert( `Hello, ${isim}!` ); // Merhaba Ahmet!

// ifade gömme

alert( `sonuç : ${1 + 2}` ); //sonuç :  3
```

${...} içerisinde yazılan ifade çalıştığında karakter dizisinin bir parçası olur. ${...} içerisine her şeyi koyabiliriz: değişken ismi adi veya matematiksel ifade 1+2 gibi.

Lütfen unutmayın ki bunu sadece kesme tırnak “`” ile yapabilirsiniz.

```
alert( "sonuç ${1 + 2}" ); // örneğin burada normal çift tırnak kullanıldığından ekrana "sonuç ${1+2}" diye çıktı verir.
```

Karakter dizileri konusunu Karakter Dizisi - Strings bölümünde daha derinlemesine incelenecektir.

Karakter tipi diye bir tip yoktur.

Bazı dillerde “character” – Karakter adında sadece bir karakteri tutan veri tipleri mevcuttur. Bu tip Java ve C’de char olarak tanımlanır.

JavaScript’te böyle bir tip bulunmamaktadır. Tek karakterli değişken de karakter dizisidir (string). Karakter dizisi bir veya birden fazla karakteri tutar.

3. Boolean (doğru/yanlış) tipi

Boolean tipi true ve false olmak üzere sadece iki değer tutabilir.

Genelde bu tip veriler doğru – yanlış sorularını tutmak için kullanılır. true doğru demek false ise yanlış demektir.

Örneğin:

```
let isimKontrolu = true; // İsim kontrolü yapıldı.

let yasKontrolu = false; // Yaş kontrolü yapılmadı.
```

Ayrıca karşılaştırma sonuçları boolean verir.

```
let buyuk = 4 > 1;

alert( buyuk ); // true (cevap görüldüğü gibi "doğru" çıkacaktır.)
```

Boolean ve diğer mantıksal işlemler Mantıksal Operatörler bölümünde daha derinlemesine incelenecektir.

“null” değeri

“null” değeri yukarıda tanımlanan hiçbir tipe dahil değildir.

Kendi başına null değerini tutar.

let yas = null;

JavaScript’te null olmayan objeyi referans göstermez veya başka dillerdeki gibi “null pointer” değildir.

“olmayan”, “boş”, “bilinmeyen değer” anlamında bir özel değerdir.

Yukarıdaki yas boş veya bilinmeyen bir değerdir.

“undefined” değeri

Bir diğer özel değer ise undefineddır. Kendi başına null gibi bir değerdir.

undefined anlam olarak “herhangi bir değer atanmamıştır” anlamına gelir.

Eğer bir değişken tanımlanmış fakat hiçbir değer atanmamışsa tam olarak bu değeri alır.

```
let x;

alert(x); // "undefined" çıktısı verir.
```

Teknik olarak undefined değerini herhangi bir değişkene atamak mümkündür:

```
let x = 123;

x = undefined;

alert(x); // "undefined"
```

Fakat bu şekilde tanımlanmasa daha iyi olur. Normalde null kullanılarak değişkenin boş veya bilinmeyen olduğu tanımlanır, undefined değişkene bir değer atanmış mı? Sorusunu kontrol eder.

4. Objeler ve Semboller

Obje özel bir tiptir.

Diğer tüm tipler “primitive” yani basit veya ilkel tiplerdir. Bu değişkenler sadece bir şey tutabilirler (karakter dizisi veya sayı). Buna karşılık objeler veri koleksiyonları (collections) veya karmaşık yapılar tutabilirler. Objeler konusunda Obje daha derinlemesine incelenecektir.

Symbol objeler için benzersiz tanımlayıcılar oluşturmak için kullanılır. Bu konuyu objeleri öğrendikten sonra öğrenmek daha iyi olacaktır.

typeof Operatörü

typeof argüman tipini bildirir. Farklı tipler için farklı akışlarınız varsa bunu kullanabilirsiniz.

İki türlü yazımı vardır:

1. Operatör olarak: typeof x.
2. Fonksiyonel tipte: typeof(x).

Diğer bir deyişle parantezli de çalışır parantez olmadan da çalışır. Sonuç aynı olacaktır.

typeof x'i çalıştırdığınızda bu fonksiyon karakter dizisi (string) döndürür:

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

Son üç satır diğerlerinden farklıdır. Şu şekilde;

1. Math matematiksel operasyonlar için kullanılacak JavaScript dilinde var olan bir objedir. Sayılar konusunda buna değinilecektir. Burada sadece objenin örneklenmesi için kullanılmıştır.
2. typeof null sonucu "object" dir. Aslında yanlış. Bu typeof fonksiyonunun bilinen bir hatasıdır. Eski versiyonlara uygunluk açısından bu şekliyle bırakılmıştır. Yoksa null bir obje değildir. Kendine has bir tiptir. Tekrar söylemek gerekirse bu JavaScript dilinin bir hatasıdır.
3. typeof alert fonksiyondur. Çünkü alert dilde doğrudan var olan bir fonksiyondur. Math ile farklı gördüğünüz gibi. Bir sonraki bölümde fonksiyonlar anlatılacaktır. Fonksiyonlar obje tipine dahildir. Fakat typeof bunları farklı yorumlar. Resmi olarak yanlış olsa da pratikte çokça kullanılan bir özelliktir.

Özet

Javascript dilinde 8 tane basit tip bulunmaktadır.

- number her türlü sayı için (integer veya floating point)
- bigint isteğe bağlı uzunluktaki tam sayılar içindir.
- string bir veya birden fazla karakter için
- boolean , true/false yani doğru-yanlış değerleri için.
- null değer atanmamış değişkenler için.
- undefined bilinmeyen değerler için.
- object daha karmaşık veri yapıları için.
- symbol eşsiz tanımlamalar için.

typeof operatörü değişkenin tipini verir.

- İki türlü kullanılabilir: typeof x veya typeof(x)
- Geriye karakter dizisi olarak değişkenin tipini döndürür. Örneğin: "string"
- İstisna olarak null kontrolünde "object" çıktısı verir. Fakat bu dile ait bir hatadır. Normalde null obje değil, kendi başına bir tiptir.

Bir sonraki bölümde basit tiplere yoğunlaşılacaktır. Bu tipleri kullanmak alışkanlık haline geldiğinde objelere geçilebilir.

Görevler

String quotes

önem: 1

What is the output of the script?

```
let name = "Ilya";

alert( `hello ${1}` ); // ?

alert( `hello ${"name"}` ); // ?

alert( `hello ${name}` ); // ?
```