25 April JavaScript

Date Types

Alttaki data tipleri primitive(ilkel) data tipledir. Stackte tutulur.(referans tipi ise heapte tutulur.) Projeye başlanıldığında const kullanılır. Değiştirilecekse let kullanılır.

Numbers ⇒ sayı kısmı için en fazla 15 virgülden sonrası için en fazla 17 basamak doğruluk garantisi vardır. Atanan her türlü sayı virgüllü normal atama hepsi number'dır. atamalarda 1_000_000 yapılabilir bu okunulabilirliği arttırır çıktıda _ görüntülenmez. NaN ⇒ not a number. ama tipi numberdır.

```
console.log(NaN == NaN); // false döner.
console.log(isNaN(k));  // eğer sayı değilse true döner.
```

```
console.log(015+025);  // başına 0 yazılırsa octal sistemde işlem yapar. outpu=34
```

strings ⇒ çift tırnak veya tek tırnak veya ` arasına alınarak tanmlanır.

```
let text2 = 'She said, "Go ahead"'; //şeklinde de atama yapılabilir
let text = "He said, \"I am a new programmer.\""; // tırnaklar aynı cins kullanılacaksa
// kaçış karakteri olarak önüne \ koyulur.
let p = `Merhaba JS ${1+2}` // bu tarz kullanım nedeninden dolayı `` tercih sebebidir.
```

booleans ⇒ false ve true.

```
""
0
-0
null
let s; => Boolean(s)
yukarıdakilerin hepsi false döner gerisi true döner.
```

Undefined ⇒ let myNumber; değişken oluşturulup atama yapılmadıysa tipi undefined olur. Hata değildir. "Is not defined" bir hatadır ve hiç tanımlanmadı ise gelir.

```
console.log(a);
var a = 3;
// bunun sebebi var ile yazılan değişken en tepeye çıkıp tanımlama yapılır. ama değer
// ataması hangi satırda yapıldıysa orada atanır. Örnekte undefined sonucu döner.
// let'te ise direk hata verir.
```

Null ⇒ boş bir değerdir.primitive'dir. bir değişkene null atanırsa çöp toplayıcılar tarafından toplanır. typeof da object döner. Bu bir bug'dır.

Bigint ve Symbol full stack için günlük hayatta fazla kullanım alanı olmadığından gösterilmedi.

https://developer.mozilla.org/en-US/docs/Glossary/Hoisting

typeof ⇒ ile type'ı öğrenilebilir.

```
let c = "2";
let d = 2;
console.log(typeof c); //string
console.log(typeof d); // number
```

console'a window yazarak window görüntülenebilir.

```
let e = prompt("Bir şey yazınız.");
console.log(typeof e);
// kullanıcıdan alınan her veri stringdir.
let e = +prompt("Bir şey yazınız.");
console.log(typeof e); // + ile number'a dönüştürülebilir.
```

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
```

```
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```


```
let l = 0.1+ 0.2;
console.log(l)
console.log(+l.toFixed(2));
//toFixed metodu ile virgülden sonra yuvarlama yapılabilir. başına artı koyunca 1 haneye
//düşer
```

Objects

JS de herşey object'dir. Referans tipidir. Birden fazla veriyi depolayabilir. Kısacası örnekteki gibi araba object'i var arabanın objeleri, özellikleri var. Anahtar- değer (key-value) çifti olarak kullanılır. Objelere fonksiyonlarda eklenebilir. Fonksiyonlara objenin method'u denir. Sırası önemli değildir.

```
const myCar = {
    make: 'Ford',
    model: 'mustang',
    year: 1965,
    color: 'Black'
    };

console.log(myCar);
myCar.color = 'Red';
console.log(myCar);
myCar.sunRoof = true;
console.log(myCar);

myCar.age = function(current) {
    console.log(current - this.year);
}

console.log(myCar);
myCar.age(2022);

delete myCar.sunRoof;
console.log((myCar));
```

Operators

bir (=) veya birden fazla değerden tek değer elde etmek için kullanılır. Aşağıdaki link operatorler için !!!!

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

Bir atama işleminde (+=, -=....) gibi kullanılırsa atama operatoru, eğer(+-/* ** %) işlemlerde kullanılırsa aritmatik operator olur. "string"+number ⇒ hata vermez birleştirir.

```
// Postfix Increment
let a = 10;
let b = a++; // a=11 b= 10
//Postfix Decrement
let a = 10;
```

```
let b = a--; //a=9 b=10
// PrefixIncrement
let a = 10;
let b = ++a; // a=11 b= 11
//Prefix Decrement
let a = 10;
let b = --a; //a=9 b=9
```

=== Strict equality ⇒ equal and of same type. Hem tipi hem değeri aynı olması lazım.

== Equality ⇒ 3==”3” true döner öncelikle tip değişikliği yapıp sonra eşitliği test ediyor.

!=eşit değilmidir. equality mantığı ile çalışır. !== ise strict equality yapısıyla aynı mantıkla çalışır.

Logical Operations

! ⇒ not operatoru tersini alıyor

&& ⇒ and operatoru Hepsi true ise son true. false var ise ilk false döner

|| ⇒ or operatoru Hepsi false ise son false, ya da ilk true döner

Öncelik sırası ⇒ ! >>> && >>> ||

?? ⇒ Nullish Coalescing Operator 1. değişken null veya undefined ise ikinci değeri atar diğer bütün durumlar için ilk değeri alır. 2 değer için çalışır sadece.

```
const nullAndString = null ?? 'Hello World'; //Hello World döner.

console.log(Boolean(null)); // false

console.log(null == false); // false
console.log(null == true); // false
console.log(null == null); // true
console.log(NaN == NaN); //false
let a,b;
console.log(a == b); //true
```

istanceof ⇒ o tipte mi ? arr instanceof Array gibi