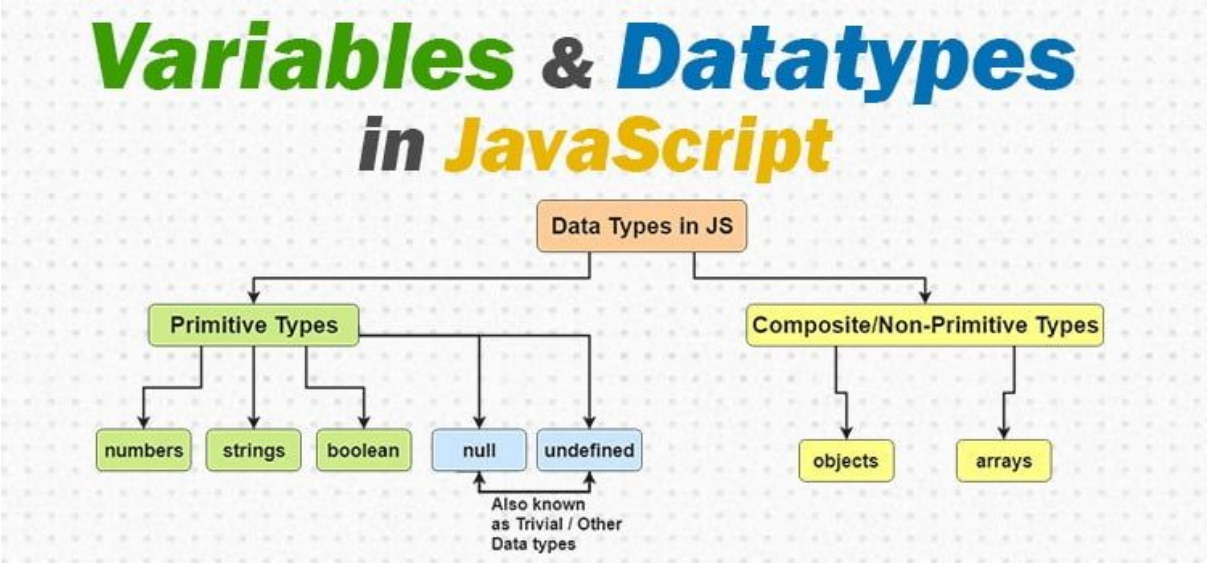
Javascript Data Yapıları

INDEX

Javascript Data Yapıları

Javascript Data Yapıları

JavaScript'te sekiz data türü vardır. Data tipi kolayca değiştirilebilir. Bu yüzden JavaScript **esnek (ya da dinamik)** veri tipine sahip bir dildir.



(https://www.bccfalna.com/number-type-in-javascript/number-type-in-javascript-in-hindi/)

1. Number

JavaScript'te sayıları göstermek için kullanılır . Kayan nokta sayıları denen ve bilimsel gösterime çok benzeyen ondalıklı üstel sayılar kullanılır. Number in matematikteki sayılardan farkı sınırlı olmasıdır. Matematikteki sayilar sonsuza kadar giderken bilgisayarda sonsuz sayı gösterilemez.
Sayılarla çarpma *, bölme /, toplama +, çıkarma -vb. islemler yapilabilir.JavaScript te kayan noktalı verilerini tamsayı olarak görüntülemek için **veri.toFixed(0)** veya **parseInt(veri)** metotları kullanılabilir. Örneğin tam sayı,kayan nokta numarası, onaltılık sayı şöyle ifade edilir.

```
var x = 2;
var y = 3.14;
var z = 0xff;
```

+Infinity sonsuzu -infinity -sonsuzu ve NaN da tanımlanmamış veya hesaplama hatasını gösterir.

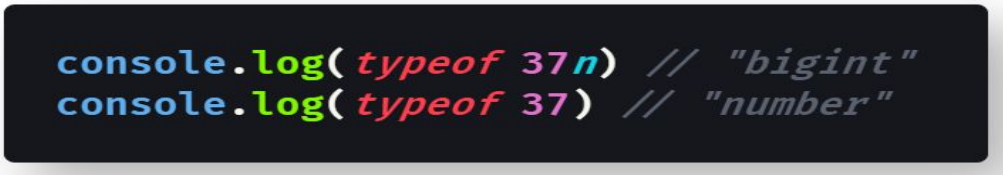
```
const number1 = 3/0;
console.log(number1); // returns Infinity

const number2 = -3/0;
console.log(number2); // returns -Infinity

// strings can't be divided by numbers
const number3 = "abc"/3;
console.log(number3);  // returns NaN
```

2. BigInt

Javascript`te(2^{53}-1)ile -(2^{53}-1) arası sayılar kullanılabilir. Daha büyük sayılar kullanmamız gerektiğinde **BigInt** kullanilir. Sayıları ve BigInts'i birlikte kullanmaya çalışmak bir TypeError ile sonuçlanır.



(https://levelup.gitconnected.com/how-to-check-if-a-number-is-a-bigint-in-javascript-6d658be6e89c

3. Boolean

Mantıksal türdür. Yalnızca evet ve hayir olmak üzere iki degeri vardir.JavaScript boş dize (""), 0, tanımsız ve null değerini yanlış olarak değerlendirir. Bunun dışındaki her şey doğrudur.

```
Boolean("true") === true
Boolean("false") === true
Boolean(-1) === true
Boolean(1) === true
Boolean(0) === false
Boolean("") === false
Boolean("1") === true
Boolean("0") === true
Boolean({}) === true
Boolean([]) === true
```

4. String

JavaScript'te **string** metin verileri için kullanılır. Bir string , harf, sayı, özel karakter ve aritmetik değerlerin birleşiminden oluşur. 3 sekilde (çift tırnak: "Hi", tek tırnak: 'Hi' veya backticks `Hi`) kullanılır. JavaScript'te çift ve tek tırnak aynı anlama gelir. Backticks `ler ifadeleri bir dizeye sararak gömmemize izin verirler ve birden fazla satır kullanabilirler.

```
let name = "John";
// embed a variable
alert( `Hello, ${name}!` ); // Hello, John!
// embed an expression
alert( `the result is ${1 + 2}` ); // the result is 3
```

5. null

Bir değişkenin değerinin olmadığını, ama daha sonra sahip olacağını belirtmek için **null** kullanilir .

```
var myVar = null; alert(myVar); // null
```

Boş değer, koşullu ifadede yanlış olarak tanımlanır. JavaScript'te null bir nesnedir.

6. undefined

Tanımlanmamış veya değeri olmayan değişken türüdür.

```
var a;
var b = "Hello World!"

alert(a) // Output: undefined
alert(b) // Output: Hello World!
```

Undefined ile null sadece tür olarak farklıdır.

```
typeof undefined          // undefined
typeof null               // object

null === undefined        // false
null == undefined         // true
```

7. Object

JavaScript'te object diğer veri türlerinden farklı olarak birden fazla veri içerebilir ve değişmez bir veriyi saklar. Object , {} parantez kullanarak oluşturulabilir.

```
var person = { firstName: "James",
               lastName: "Bond",
        age: 15, getFullName:
function () { return this.firstName + ' ' + this.lastName } }
```

new kelimesini kullanarakta nesne olusturulabilir.

```
    var person = new Object();
```

Nesnenin özelliklerine ve yöntemlerine nokta veya [] köşeli ayraç kullanılarak erişilebilir.

Bir nesne, özellik olarak başka bir nesneyi içerebilir.

```
var person = { firstName: "James", lastName: "Bond", age: 25, address: {
id: 1, country:"UK" } }; person.address.country; // returns "UK"
```

8. Symbol

Bir sembol oluşturduğunuzda her seferinde benzersiz bir sembol oluşur.Semboller genellikle nesne özelliklerini tanımlamak için kullanılır ve özellikler arasındaki isim karmaşasını önler.

```
const NAME = Symbol()const person = {[NAME]: 'Flavio'}person[NAME] //'Flavio'const RUN = Symbol()person[RUN] = () => 'Person is running'console.log(person[RUN]()) //'Person is running'
```

Ancak bazen aynı isimli sembollerin de olmasını isteriz.Bunun için **global symbol registry** i kullaniriz. İçinde semboller oluşturabilir ve onlara daha sonra erişebiliriz.
Registry den bir sembolü okumak (yoksa oluşturmak) için **Symbol.for(key)** kullanılır.
Birde JavaScript'te **"sistem"** sembolü vardır ve bunları nesnelerde ince ayar yapmak için kullanılır.
Sembollere birkaç örnek:

- **Symbol.hasInstance**
- **Symbol.isConcatSpreadable**
- **Symbol.iterator**
- **Symbol.toPrimitive** dir .

Object.getOwnPropertySymbols() yöntemini kullanarak da bir nesneye atanmış tüm simgelere erişilebilir..