26 May JS

OBJECTS

3 farklı yöntem ile object oluşturulabilir.

1. Object() class'ından new Operatörü ile

```
const araclar = new Object();
araclar.marka = "BMW";
araclar.moto = 1.3;
araclar.model = 2022;

console.log(araclar);
```

2. Object constructor'ı kullanarak

```
//* Object constructor
function Personel(id, ad, maas) {
    this.perId = id;
    this.perAdi = ad;
    this.perMaas = maas;
    console.log(this); //! Personel objesine bağlanmıştır (binded)
  }

  const ahmet = new Personel(101, "Ahmet", 75000);
  const canan = new Personel(102, "Canan", 85000);
  console.log(ahmet, canan);
  console.log(canan.perMaas);
  console.log(ahmet.perAdi);
  console.log(this); //! window objesine bağlanmıştır
```

3. Object Literal (En çok tercih edilen yöntem)

```
const calisan = {
  ad: "ahmet",
  soyad: "Yilmaz",
  yas: 30,
  is: "developer",
  diller: ["C", "C++", "Python", "JS"],
  maas: 120000,
};
```

const isci = calisan; direk atama yapılması referans aktarım denir. Aslında referansı atanmış oluyor ve birinde yapılan değişiklik diğerinin değerini de değiştirir.

Object Metotları

Objelere, classlara ait olan fonksiyonlara medot denir.

arrow functionlar `this` erişimine sahip değildir. Arrow function'ların global scope u göstermesi gerekir.

```
//! NOT: arrow fonksiyonları farklı amaç için geliştirilmiş fonksiyonlarıdır
//! ve lexical context'e sahiptirler. Dolayısıyla, bir arrow fonk. içerisinde
//! this kelimesi kullanılrsak beklenmeyen sonuclar alabiliriz.
//! Çünkü, arrow içerisindeki this kelimesi global scope'u gösterir. (window nesnesini) gösterir.
//! Bunu engellemek için object fonskyionlarını tanımlamak için normal fonksiyon  yöntemlerini kullanmak gerekir.
```

```
const kisi = {
  ad: "Can",
  soyad: "Canan",
  dogum: 1990,
  meslek: "developer",
  ehliyet: true,
  yasHesapla: function () {
    return new Date().getFullYear() - this.dogum;
  },
  ozet: function () {
    return `${this.ad}, ${this.yasHesapla()} yasindadir`;
  },
};
```

https://flaviocopes.com/javascript-this/

Object Iteration

Doğrudan itere edilebilir dizi içerisine nesneleri koymak iterasyon için uygundur ve kolaydır.

JSON => javascript Object Notation.

NEW GENERATION OPERATORS: DESTRUCTURING (OBJECT)

https://flaviocopes.com/javascript-destructuring/

```
const car = {
  name: "BMW",
  model: 1990,
  engine: 1.6,
};
//* 1.YONTEM (Classical)
console.log(car.model);
console.log(car["name"]);

// 2. YONTEM : DESTRUCTURING
const { name, model } = car;
console.log(name, model); // bu yöntemde tanımlamada aynı isimler olması gerekiyor.
const { name: c2Name, model: c2Model } = car2;
console.log(c2Name, c2Model); // bu tarzda isim değişikliği yapılabilir.
```

NEW GENERATION OPERATORS: DESTRUCTURING (ARRAY)

Daha az kullanışlı… iki virgül arası boş verilerek eleman atlanılabilir.

```
// NEW GENERATION OPERATORS: DESTRUCTURING (ARRAY)

const names = ["Ahmet", "Mehmet", "İsmet", "Saffet"];

const name1 = names[0];
```

```
const name2 = names[1];

const [person1, person2, , person4] = names;
console.log(person1, person2, person4);
```

NEW GENERATION OPERATORS: SPREAD OR REST (...)

```
/* REST: (Arrays) */
const vehicles = ["bmw", "reno", "mercedes", "ferrari", "anadol"];

const [vec1, vec2, ...restVehicles] = vehicles;
```

```
const personel = {
  pName: "john",
  surname: "smith",
  job: "developer",
  age: 30,
};

const { pName, job, ...surnameAge } = personel;
// job be pName i ayrı alıyor geri kalan hepsini array a atıyor.
```

```
// SPREAD

const araclar = ["Ucak", "Helikopter", "Bisiklet"];
const otomobiller = ["Tır", "Otobus", "Araba", "SUV"];

const tumAraclar = [...araclar, ...otomobiller];
console.log(tumAraclar);
```