JavaScript Ders Notları - 2

Diziler

Birden fazla değeri bellekte saklamaya yarayan değişkenlerdir. Değişken veya sabit olarak tanımlanabilir, köşeli parantez içinde virgülle ayrılarak tanımalanır ve tüm değişken tanımlama kuralları geçerlidir.

```
var meyveler = [“elma”, “portakal”, “çilek]
let sayilar = [1, 2, 3, 4, 5]
```

Dizi içindeki değerlere erişim sağlamak için indis numaralarından yararlanılır. İndis dizi içindeki değerlerin sıra numarası olarak düşünülebilir.

console.log(meyveler[0]) // elma

console.log(sayilar[3]) // 4

- Not 1: Dizide bulunan verilerin aynı türde olma zorunluluğu yoktur.

  ```
  let mix = [“Eskişehir”, 26, 222, true]
  ```
  şeklinde farklı veri türleri aynı dizi içinde yer alabilir.

- Not 2: Dizinin elemanı da dizi olabilir.

  ```
  let eskisehir = [“Eskişehir”, 26, 222, [“Tepabaşı”, “Odunpazarı”, “Çifteler”, “Seyitgazi”]]
  ```

Nesneler (Object)

Nesne tabanlı programlama yaklaşımının temelini oluşturan bu yapılar da aslında dizilere benzer bir değişken depolama birimleridir. Dizilerden farklı tarafı elamanların indisleri 0, 1, 2… şeklinde gitmez bizim tarafımızdan belirlenir. {} parantez içinde tanımlama yapılır. İçinde farklı türde veriler bulunabilir, ayrıca nesne içinde nesne ve nesne içinde fonksiyon tanımlaması yapılabilir. Nesne içinde bulunan fonksiyonlar metot olarak isimlendirilir.

```
let Turkiye = {
    “ankara”: “06”,
    “istanbul”: “34”,
    “eskisehir”: 26
}

let öğrenci = {
    “isim”: “Ahmet”,
    “sinif”: 11,
    “sube”: “A”,
    “dogumYili”: 2005,
    “yas”: function(){
        return 2022 – this.dogumYili
    }
}
```

Yukarıda tanımlanan Turkiye nesnesinin oldukça basit bir yapıda olduğunu görebilirsiniz nesnenin içerdiği değerlere (property olarak isimlendirilir) erişim şu şeklinde sağlanır.

```
console.log(Turkiye.ankara) // 06
```

öğrenci nesnesi ise içinde bulunan yas metodu ve this anahtar kelimesi sebebiyle biraz daha karmaşık olduğu söylenebilir. **This** anahtar kelimesi nesnenin kendisini işaret eder ve nesnenin bir property si içinde kendisine ait farklı propertylerine erişmek için kullanılır ve oldukça pratik bir yaklaşımdır.

```
Console.log(ogrenci.isim) // Ahmet
Console.log(ogrenci.yas()) // 17
```

Şeklinde çıktı verecektir, fakat metot kullanımına dikkat ederseniz parantez açıp kapatmak gerekmektedir çünkü bu bir fonksiyondur ve çalışması için onu çağırmanın kuralı budur.

Nesnelerin propertylerine yeni değerler atamak için aynı yöntem kullanılır:

```
Ogrenci.dogumYili = 2004
Ogrenci.dogumYeri = “Eskişehir”
```

Extends (Nesne Kopyalama)

Nesne tabanlı programlama yaklaşımının temelinde if/else gibi kontrol değimlerinin azaltılması ve daha düzenli bir yazılım geliştirme süreci bulunmaktadır. Bu yüzden nesne tanımlamaları genel yapılır ve yeni nesneler bu genel nesneden türetilir. Örneğin farklı türde arabaların özelliklerini değişkenler içinde mi tutmak mantıklı olur, diziler içinde mi tutmak mantıklı olur yoksa bir nesne tanımlayıp tüm arabaları bu nesneden türetmek mi mantıklı olur sorusuna dilerseniz yüzlerce arabanın yüzlerce özelliği için binlerce değişken tanımlayabilirsiniz ☺. Bu örnekten yola çıkacak olursak bir tane araba nesnesi tanımlayalım ve tüm arabaları bu nesneden türetelim:

```
    const araba = {
        "marka": "",
        "model": "",
        "motor": "",
        "yakit": "",
        "vites": "",
        "renk": "",
        "ortYakit": 0,
        "neYakar": function(km, yakitFiyati){
            return (km * this.ortYakit / 100) * yakitFiyati
        }
    }
```

Gördüğünüz gibi nesnenin tüm propertyleri boş bırakılarak oluşturuldu yeni bir araba tanımlamak için:

```
let bmw3 = new araba();
```

Aynı özelliklerde bir nesne kopyalandı (extend edildi) artık özelliklerini girebiliriz.

```
    bmw3.marka = “BMW”;
    bmw3.model = “3”;
    bmw3.motor = “1.6”;
    bmw3.yakit = “dizel”;
    bmw3.vites = “otomatik”;
    bmw3.renk = “Kırmızı”;
    bmw3.ortYakit = “7”;
```

Constructor (Yapıcı Metot)

Fakat burada şöyle bir problem görülebilir, genel olan araba nesnesi tanımlanırken tüm özellikleri boş bırakıldı bu istenen bir durum değildir ve nesneye varsayılan değer atanması oldukça önemlidir bu varsayılan değerleri atayan metot constructor (yapıcı) metot olarak isimlendirilir. Kullanımı şu şekildedir:

```
    let araba = {};
        function araba(marka, motor, yakit, vites, ort){  //
constructor (yapıcı) metot
        this.marka = marka,
        this.motor = motor,
        this.yakit = yakit,
        this.vites = vites,
        this.ort = ort,
        this.tuketim = (km) => {
            return km * this.ort / 100;
        }
    };
```