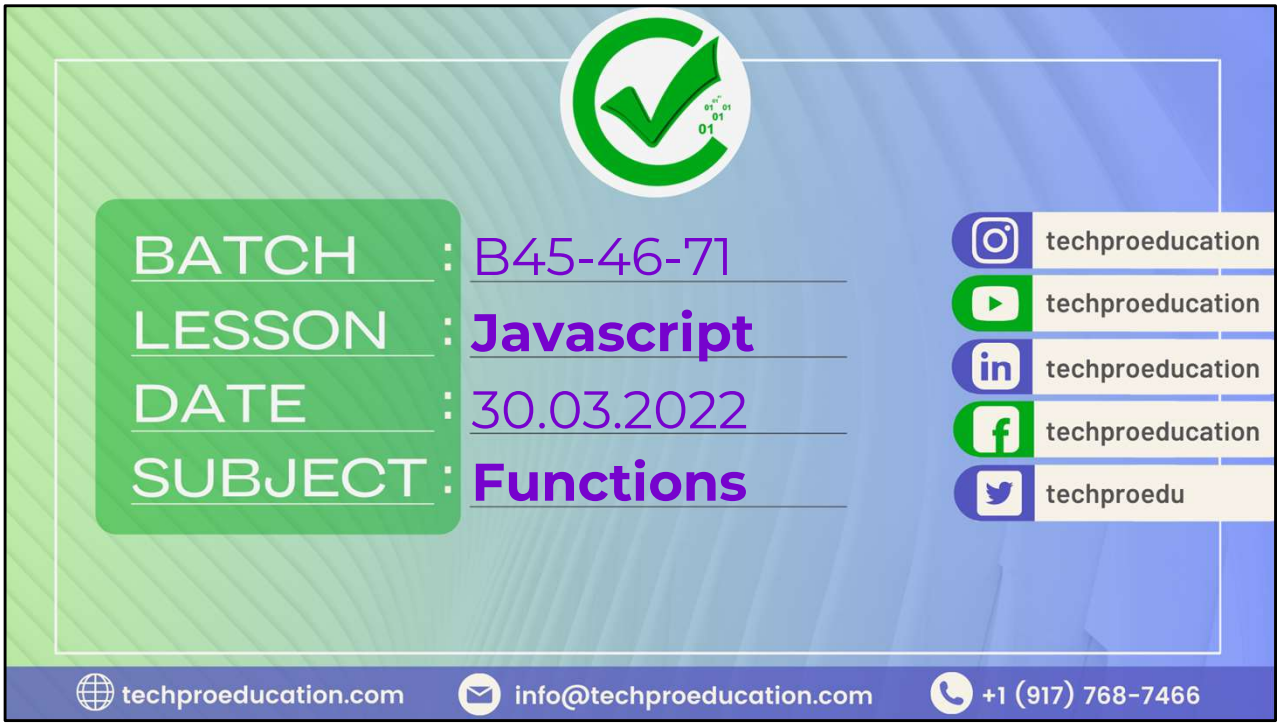

BATCH : B45-46-71
LESSON : Javascript
DATE : 30.03.2022
SUBJECT : Functions
techproeducation
techproeducation
techproeducation
techproeducation
techproedu
techproeducation.com
info@techproeducation.com
+1 (917) 768-7466

Fonksiyon tanımlama

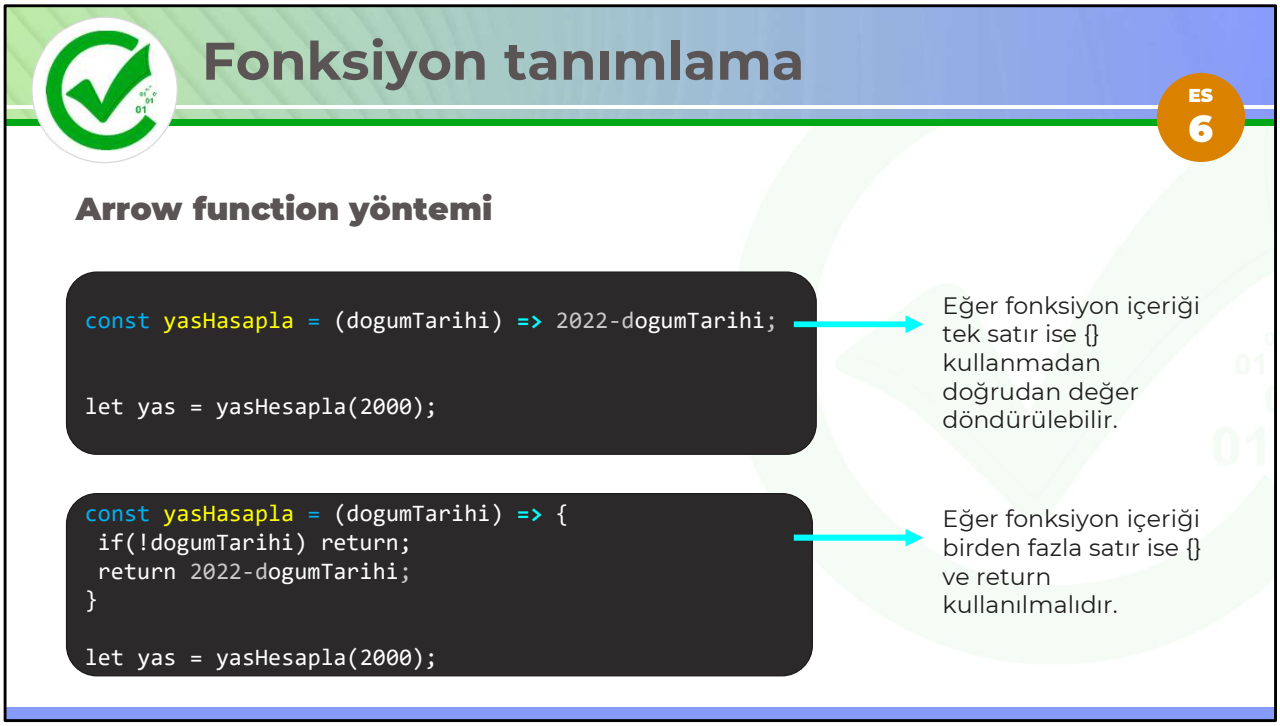
Expression yöntemi

```
const yasHasapla = function(dogumTarihi){
    return 2022 - dogumTarihi;
}

yasHesapla();
```

- Javascript'te fonksiyonlar ifade (**expression**) olarak da tanımlanabilmektedir.
- Bu yöntemde, fonksiyonlar **isimsizdir** (**anonymous**) ve bir **değişkene atanırlar**. Dolayısıyla fonksiyonun bir dönüş değeri olmalıdır.

- Bu değişken, fonksiyon olarak kullanılır.
- Bu yöntemde fonksiyon tanımlanmadan önce çağrılırsa JS hata verecektir. Dolayısıyla **expression** yöntemini kullanmak için önce fonksiyonu tanımlamak sonra çağırmak gerekir.
- Programcıyı, önce fonksiyonların tanımlanması, sonra kullanılmasına zorladığı için aslında daha düzenli ve daha anlaşılır kod yazmaya olanak sağlamaktadır.
- Fonksiyonların ve değerlerin değişkenlerde saklanmasını gerektirmektedir. Bu da daha sade bir kodlama demektir



Arrow functions, fonksiyonları tanımlamanın en kısa yoludur

Fonksiyon tanımlama

Arrow function

```
const topla = (a, b) => a+b;
```

Expression

```
const topla = function(a, b){
    return a+b;
}
```

Geleneksel

```
function topla(a, b){
    return a+b;
}
```

Fonksiyon

PRACTISE

Textbox a girilen sayının, sayı olup olmadığını kontrol ettikten sonra, faktöriyelini hesaplayıp sonucu döndüren fonksiyonu arrow function yöntemi ile yapınız.


Scope
Değişken ve sabitlerin yaşam alanına scope denir
function scope
Fonksiyon içinde tanımlanan değişkenler sadece fonksiyon içinde geçerlidir.
global scope
Her yerden erişilebilen değişkenler, en tepede tanımlanır.
block scope
Sadece tanımlandığı blok içinde (if, for, while...) geçerlidir.

Function Scope

```
const fonk1 = function () {
    let sayi1 = 22;
    console.log(sayi1);
};

fonk1();
console.log(++sayi1);
```

sayi1 değişkeni fonk1 içinde tanımlandığı için, sadece fonk1 fonksiyonu içinde geçerlidir. Fonksiyon dışından erişilmeye çalışıldığında hata alınır.

Global Scope

```
let sayi = 5;
const fonk = function (){
    sayi = 10;
    console.log(`Fonk. İçi: ${sayi}`);
}

fonk();
console.log(`Fonk. Dışı: ${++sayi}`);
```

```
Fonk. İçi: 10
Fonk. Dışı: 11
```

sayi değişkeni fonk dışında tanımlandığı için, her yerden erişilebilir. Örnekte hem fonk isimli fonksiyondan hem de dışındaki kodlar üzerinden erişilebilmektedir.

Global vs Function Scope

```
let sayi = 3;
const fonk = function () {
   let sayi = 7;
   console.log(`Fonk. İçi: ${sayi}`);
};

fonk();
console.log(`Fonk. Dışı: ${++sayi}`);
```

```
Fonk. İçi: 7
Fonk. Dışı: 4
```

Hem global he de function scope da aynı isimde değişken tanımlandığında bunları farklı değişkenler olarak algılanır.

Block Scope

ES 6

```
const fonk = function (sayi) {
    if (sayi < 0){
        let negatif = true;
    }
    console.log(negatif);
};
fonk(-4);
```

Block scope değişkenler sadece bulundukları { } blok içinde geçerlidir.
Blok dışından erişim yapılamaz.
Block scope sadece let ve const için geçerlidir.

var vs let

var ile **let** arasındaki en önemli fark, scope farklılığıdır. **var** function scope, **let** block scope olarak davranır.

```
function test(){
    var a = "Merhaba";
    let b = "Dünya";
    if(true){
        var x = "Ankara";
        let y = "İzmir";
    }
    console.log(a, b, x);
    console.log(y);
}
```

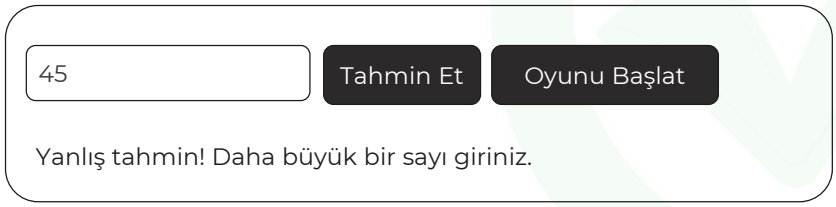


```
Merhaba Dünya Ankara
▸Uncaught ReferenceError: y is not defined
```

Scope

PRACTISE

«Oyuna Başla» butonuna basıldığında 1-100 arasında rasgele sayı tutulsun. Textbox içine tahmini sayı girilip «tahmin et» butonuna basıldığında tutulan sayıyla girilen sayıyı kontrol edip doğru tahmin olup olmadığını ve büyüklük küçüklük durumunu gösteren programı yapınız.



İpucu `let sayi = Math.floor(Math.random() * (max - min + 1)) + min`