

## **1. Giriş**

Merhaba ben İsa Sarı. Patika.dev ve Protein tarafından organize edilen Vue.js Bootcamp'ine teknik test ve mülakat sürecinden sonra kabul aldım. Bu doküman serisinde, bu bootcamp boyunca derslerden aldığım notlarımı paylaşacağımı belirtmişim. Yine tekrar belirtmek gerekirse, bu dokümanda bootcampte aldığım notların yanı sıra kendi yaptığım eklemeler de yer alacak. Dokümanda karşılaşılabilecek yanlışlıklar veya eksiklikler tamamen bana aittir. Geribildirim için iletişim kurmaktan çekinmeyin. Mail adresim: [dev.isasari@yandex.com](mailto:dev.isasari@yandex.com) .



*Bu doküman, Creative Commons Attribution-NonCommercial-NoDerivs 2.0 ile lisanslanmıştır.*

## **İçindekiler**

1. Giriş.....	1
2. JavaScript - Fonksiyonlar .....	3
2.1. Fonksiyon nedir? .....	3
2.2. "Klasik Fonksiyon"un Yapısı .....	3
2.3. Nasıl "Klasik Fonksiyon" Oluştururuz? .....	4
2.4. Fonksiyon İsmi Nasıl Belirlenmeli? .....	5
2.5. Fonksiyon Nasıl Çağrılır? .....	5
2.6. Fonksiyona "return" Ekleme.....	6
2.7. Parametre ve Argüman Nedir? .....	6
2.8. Parametreler ve Argümanlarla İlgili Detaylı Bilgiler .....	9
2.9. Fonksiyonun İçinde Fonksiyon Oluşturmak.....	13
2.10. Klasik Anonim Fonksiyon Yapısı .....	14
2.11. Anonim Fonksiyon ve Normal Fonksiyon Arasındaki Farklar.....	16
2.12. Modern Fonksiyon Yapısı (Arrow Fonksiyon) .....	18
2.13. Fonksiyonun Çalıştırılma Zamanı Nasıl Ayarlanır? .....	22
2.14. Fonksiyonlar Konusunun Devamı .....	23
3. JavaScript'te Diğer Konular.....	23
4. GIT Versiyon Kontrol Sistemi Nedir? .....	24
5. Chrome DevTools nedir? .....	24
6. ViteJS nedir? .....	25
7. Vue.js'e Giriş .....	25
8. Yararlı Linkler / Kaynaklar .....	26

## 2. JavaScript - Fonksiyonlar

Bootcamp derslerinde fonksiyonlarla ilgili temel bilgilerden ziyade “best practice”leri işledik. Ancak ben bu dokümanımda, fonksiyonların temellerinden başlayıp daha sonra best practicelere geçeceğim.

Bu konuyu anlatırken formasyon eğitimim sırasında edindiğim öğretim yöntem ve tekniklerini uygulayacağım.

### 2.1. Fonksiyon nedir?


Fonksiyon, belirli bir görevi gerçekleştirmek için tasarlanmış bir kod bloğudur.

Neden fonksiyona ihtiyaç duyarız? Çünkü çoğu zaman kod yazarken belirli bölümleri tekrarlama ihtiyacı duyulur. Fonksiyon, bizi tekrar tekrar aynı kodu yazmaktan kurtarır. Çünkü fonksiyonu bir kere tanımladıktan sonra istediğimiz kadar kullanabiliriz. Aynı fonksiyonla farklı argümanlar ve farklı sonuçlar elde edebiliriz. Ayrıca fonksiyon, kod kalabalığını engeller daha az ve okunabilir kodlar yazmamızı sağlar.

JavaScript fonksiyonlarını gelişmişlik açısından ikiye ayırarak inceleyebiliriz: Klasik Fonksiyon ve Modern Fonksiyon. Bu dokümanda, öncelikle “Klasik Fonksiyon” yapısı, daha sonra ECMAScript 6 ile gelen “Arrow Fonksiyon” veya bizim deyimimizle “Modern Fonksiyon” yapısı incelenecektir.

### 2.2. “Klasik Fonksiyon”un Yapısı

Bir fonksiyonun en temel yapısı şu şekildedir:



```
1  function functionName() {  
2    // code  
3  }  
4  
5  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

### 2.3. Nasıl “Klasik Fonksiyon” Oluştururuz?

Bir fonksiyon oluşturmak için önce “function” kelimesi kullanılır, daha sonra fonksiyon adı kullanılır ve normal parantez açılıp kapatılır ve sonrasında süslü parantez açılıp kapatılır. Bunlar bir fonksiyonun temel yapıtaşlarıdır.

Fonksiyon adından önce gelen “function” kelimesiyle yazdığımız kod bloğunun fonksiyon olduğunu belirtmiş oluruz.

Fonksiyona isim verirken, değişken ismi yazma kurallarına uyulması gerekir.

Eğer parametreler kullanılacaksa normal parantez içerisine yazılır ve virgüllerle ayrılırlar. Bir fonksiyona parametre eklemesek bile normal parantezi yazmamız gerekir. Parametreler ve argümanlar birbiri yerine kullanılan kelimeler olmasına rağmen aralarında önemli bir fark bulunur. Parametreler, tanımladığımız fonksiyondaki normal parantezler arasında yer alan ifadelerken; argümanlar, fonksiyon çağrıldığı zaman aldığı değerlerdir.

Kodlarımızı süslü parantez içerisine yazarız. Bu kısma fonksiyonun gövdesi, kod bloğu, kapsam ya da scope denilir.

Örneğin siteye girildiğinde ekrana “Hoş Geldiniz!” yazısını getiren bir fonksiyon oluşturalım.

A code editor window with a dark background and light-colored text. It shows a JavaScript function definition. The code is as follows:

```
1 function showMessage() {  
2   alert('Hoş Geldiniz!');  
3 }  
4  
5 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Ancak bu fonksiyonumuz çalışmayacaktır. Çünkü fonksiyonumuzu çağırmadık.

## **2.4. Fonksiyon İsmi Nasıl Belirlenmeli?**

Fonksiyon bir işi yerine getiren kod parçasıdır. Fonksiyona isim koyarken önce hangi işi/eylemi yaptığını belirten bir fiil seçilmeli sonra da o fiilin nesnesini ardına eklemeliyiz.

Örneğin sandviç yapan bir fonksiyonumuz olsun. Bu fonksiyonumuza isim koyarken önce fonksiyonun yaptığı işi belirten bir fiil seçeriz: make. Sonra da yaptığı şeyin ne olduğunu ekleriz: Sandwich. Dolayısıyla sandviç yapan bir fonksiyon için fonksiyon adının "makeSandwich" olması iyi bir seçim olacaktır. Çünkü fonksiyonun adına bakıldığında ne yaptığı açıktır.

Fonksiyonlara isim verirken Türkçe yerine İngilizce tercih edilmelidir.

Fonksiyon isimlerinde, camelCase yazım şekli tercih edilmelidir. Çünkü fonksiyon isimlendirmede genellikle camelCase tercih edilir.

## **2.5. Fonksiyon Nasıl Çağrılır?**

Bir JavaScript fonksiyonunu, oluşturduktan sonra, çalıştırmak için onu çağırmamız gerekir. Bunun için birçok yöntem var. Birincisi bir tetikleyici oluşturarak (örneğin butona basıldığında), ikincisi bir JavaScript kodu içerisine ekleyerek ve üçüncüsü otomatik olarak çalıştırabiliriz.

Örneğin yukarıda yazdığımız fonksiyonumuzu şöyle çağırıp çalıştırabiliriz:



```
1 function showMessage() {  
2   alert('Hoş Geldiniz!');  
3 }  
4  
5 showMessage();  
6  
7 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

## 2.6. Fonksiyona “return” Eklemek

Bazı durumlarda fonksiyona “return” ifadesini eklemek gerekebilir.

Örneğin iki sayıyı birbiriyle çarpan bir fonksiyon oluşturalım. Böyle bir fonksiyonda return kullanmak gerekmektedir. Çünkü fonksiyon içinde yapılan çarpma işleminin sonucunda çıkan değeri dışarıya aktarmamız gerekir. Bunu “return” ile sağlarız.

```
1 function multiplyNumbers(a, b) {  
2   return a * b;  
3 }  
4  
5 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Return ifadesi, fonksiyonumuzun çalışmasını durdurmak için de kullanılabilir. Eğer fonksiyonumuzun çalışmasını durdurmamız gereken bir senaryo varsa bu durumda da return kullanmak gerekir.

Fonksiyon içinde return görülen yerde fonksiyondan çıkılır. Bundan dolayı return’ü fonksiyonun bitişini ifade etmek için de kullanabiliriz. Uzun kodlar ve içe içe fonksiyon içeren fonksiyonlarda, kodun okunabilir olmasını sağlamak için return kullanmak önem taşır.

Fonksiyonlar her zaman bir şey döndürürler. Eğer return kelimesi yoksa sonuçta yine de “undefined” döner.

## 2.7. Parametre ve Argüman Nedir?

Parametre ve argüman sıklıkla birbiri yerine kullanılmakta ve bazen de karıştırılmaktadır. Ancak aralarında önemli bir fark bulunur. Bu farkı anlamak için parametre ve argümanın ne olduğunu “bir sandviç yapma” örneği üzerinden anlatacağım. Bu örnek, aynı zamanda, fonksiyonun daha iyi anlaşılmasını da sağlayacak.

- 1- Bir sandviç yaparken
- 2- Bazı malzemelere (ekmek, peynir, sebze) ihtiyaç duyarız,
- 3- Sonra bu malzemeleri bir araya getiririz;
- 4- Bu işlem sonucunda sandviç elde etmiş oluruz.

## İsa Sarı – Protein & Patika.dev Vue.js Bootcamp Notlarım – 2. Hafta

Bu örnekte, "sandviç yapmak" fonksiyonumuzun adıdır. Sandviç malzemelerimiz olan ekmek, et ve sebze fonksiyonumuzun parametreleridir. Bu malzemeleri bir araya getiririz, yani sandviç yaparız. Bu işlem sonucunda sandviç elde etmiş oluruz. Yani bunları fonksiyonda kullanılabilecek şekilde ifade edelim:

- 1- Bir sandviç yaparken → makeSandwich
- 2- Bazı malzemelere (ekmek, peynir, sebze) ihtiyaç duyarız → (bread, cheese, vegetable)
- 3- Sonra bu malzemeleri bir araya getiririz → bread + cheese + vegetable
- 4- Bu işlem sonucunda sandviç elde etmiş oluruz → sandwich

Şimdi de fonksiyon oluşturma kurallarımıza göre "bir sandviç yapma" fonksiyonumuzu oluşturalım:

```
1 function makeSandwich(bread, cheese, vegetable) {  
2   let sandwich = bread + cheese + vegetable;  
3   return sandwich;  
4 }  
5  
6 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Şimdi elimizde bir sandviç yapma fonksiyonumuz var. Bu fonksiyonumuzla farklı sandviçler hazırlayabiliriz. Örneğin bazıları sandviç ekmeğini tam tahıllı isterken bazıları glutensiz ekmek bazıları kepek ekmeği isteyebilir. Bunların yanı sıra daha birçok ekmek çeşidi ve bunları sevenler var. Ayrıca bazıları peynir olarak ezine peynirini severken bazıları cheddar peynirini bazıları dil peynirini bazıları ise hellim peynirini sevebilir. Bunların yanı sıra daha birçok peynir çeşidi ve bunları sevenler var. Sebze olarak bazıları roka severken bazıları kırmızı biber bazıları da domatesi sevebilir. Biz sandviç yapma fonksiyonumuz ve içinde bulunan parametreler sayesinde istediğimiz zaman istediğimiz ekmek, peynir ve sebze ile istediğimiz veya istenilen sandviçi yapabileceğiz. Dolayısıyla bu örnekten yola çıkarak parametrelerin ekmek, peynir, sebze; argümanların ise ekmek çeşidi, peynir çeşidi ve sebze çeşidi olduğunu söyleyebiliriz.

## İsa Sarı – Protein & Patika.dev Vue.js Bootcamp Notlarım – 2. Hafta

Örneğin ekmek olarak tam tahıllı ekmek, peynir olarak cheddar peyniri, sebze olarak domates içeren cheddarlı sandviçimizi yapalım:

```
1 function makeSandwich(bread, cheese, vegetable) {
2   let sandwich = bread + cheese + vegetable;
3   return sandwich;
4 }
5
6 makeSandwich("tam tahıllı ekmek", "cheddar", "domates");
7
8 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Fonksiyonumuzu console.log ile yazdırdığımızda cheddarlı sandviçimiz hazır olmuş olacak. Afiyet olsun!

İşte burada, tam tahıllı ekmek, cheddar ve domates "argüman"larımızı oluşturmuş oluyor.

Bunları bir de şekil üzerinde gösterelim:

```
1
2 function makeSandwich(bread, cheese, vegetable) {
3   let sandwich = bread + cheese + vegetable;
4   return sandwich;
5 }
6
7
8 makeSandwich("tam tahıllı ekmek", "cheddar", "domates");
9
10
11 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

parametreler  
↓  
argümanlar  
↓



## İsa Sarı – Protein & Patika.dev Vue.js Bootcamp Notlarım – 2. Hafta

Şimdi daha düzenli, temiz ve okunabilir kod yazımı için yani başka bir deyişle best practicesi uygulamak adına sandviçime isim veriyorum. Sandviçimde kullanacağım her bir malzemeye değişken atıyorum. Sonra da fonksiyonumu kullanarak sandviçimi hazırlıyorum:

```
1 function makeSandwich(bread, cheese, vegetable) {
2   let sandwich = bread + cheese + vegetable;
3   return sandwich;
4 }
5
6 let breadOne = "tam tahıllı ekmek";
7 let cheeseOne = "cheddar";
8 let vegetableOne = "domates";
9
10 let mySandwich = makeSandwich(breadOne, cheeseOne, vegetableOne);
11 console.log(mySandwich);
12
13 // "tam tahıllı ekmekcheddardomates"
14
15 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Sandviç fonksiyonu örneğimizi gördükten sonra artık parametre ve argümanın ne olduğunu tanımlayabilir, aralarındaki farkı söyleyebiliriz. Parametreler, fonksiyonumuzu oluştururken yazdığımız girdilerdir. Argümanlar ise fonksiyonumuzu çağırırken girdiğimiz verilerdir. Başka bir deyişle, argümanlar, fonksiyon çağırılırken parametrelere aktarılan değerlerdir. Dolayısıyla aralarındaki fark da şudur: Parametreler fonksiyon oluştururken yazılır, argümanlar fonksiyon çağırılırken yazılır.

### 2.8. Parametreler ve Argümanlarla İlgili Detaylı Bilgiler

JavaScript fonksiyonlarında;

- ✓ Parametre girmek zorunlu değildir. Örneğin aşağıdaki fonksiyonda web sayfasına girildiğinde alert box çıksın ancak parametre ve argüman girmedik için hiçbir mesaj iletmesin. Bu fonksiyonumuz parametre içermemesine rağmen hatasız çalışacaktır:

```
1 function showMessage() {  
2   alert();  
3 }  
4  
5 showMessage();  
6  
7 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

- ✓ Parametre girdiğimiz bir fonksiyonu çağırırken argüman aktarma zorunluluğu yoktur. Örneğin şöyle bir fonksiyon oluşturalım; kullanıcı, web sitesine girdiğinde hemen bir alert çıksın ancak herhangi bir mesaj bulunmasın. Bu fonksiyona bir mesaj (text) parametresi ekleyelim. Ancak fonksiyonu çağırırken bu parametreye argüman aktarmayalım. Fonksiyonumuz çağırdığımızda argüman girmemiş olmamıza rağmen çalışacaktır. Fakat argüman girmediklerimiz için "undefined" yazacaktır:

```
1 function showMessage(text) {  
2   alert(text);  
3 }  
4  
5 showMessage(); // ekrana "undefined" gelir  
6  
7 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

- ✓ Birçok parametre girdiğimiz bir fonksiyonun her bir parametresi için argüman aktarma zorunluluğu yoktur. Örneğin sandviç yapma fonksiyonumuzda, eğer

sandviçimize sebze koymak istemiyorsak o halde, bir sebze parametresi yerine bir argüman girmeden de sandviçimizi hazırlayabiliriz:

```
1 function makeSandwich(bread, cheese, vegetable) {  
2   let sandwich = bread + cheese + vegetable;  
3   return sandwich;  
4 }  
5  
6 makeSandwich("tam tahıllı ekmek", "cheddar");  
7  
8 // tam tahıllı ekmekcheddarundefined  
9  
10 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Ancak fonksiyonumuzu console.log ile yazırdığımızda sebze parametresine bir sebze türü yani argüman aktarmadığımız için "undefined" olarak döndü. Demek ki, parametre girilip argüman girilmediği taktirde bize default olarak "undefined" dönecektir.

- ✓ Parametre veya parametrelere varsayılan (default) olarak değer girebiliriz. Böyle bir durumda, parametre sayısı kadar argüman aktarmazsak varsayılan değer, argüman yerine geçer. Sandviç fonksiyonumuzla bunu örnekleyelim. Ben sandviçimde sebze olarak hep domates olsun fakat ekmek ve peynir çeşitleri değişsin istiyorum. Bu durumda, sandviç fonksiyonumuzun sebze parametresi kısmında "domates"i varsayılan (default) olarak girebilirim. Böylece ilgili parametreye argüman aktarmasam bile "domates" default olarak gelecektir.



```
1 function makeSandwich(bread, cheese, vegetable= "domates") {  
2   let sandwich = bread + cheese + vegetable;  
3   return sandwich;  
4 }  
5  
6 makeSandwich("tam tahıllı ekmek", "cheddar");  
7 // tam tahıllı ekmekcheddardomates  
8  
9 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Ancak argümanı aktardığımız taktirde varsayılan değer değil aktarılan argümanı çıktı alırız. Yani örneğimizden devam edecek olursak eğer, ben "sebze" parametreme "domates"i default olarak yazmış olmama rağmen canım domates istemediği için kapyra biber ile sandviç yapmak istiyorsam "kapyra biber"i argüman olarak aktarabilirim:



```
1 function makeSandwich(bread, cheese, vegetable= "domates") {  
2   let sandwich = bread + cheese + vegetable;  
3   return sandwich;  
4 }  
5  
6 makeSandwich("tam tahıllı ekmek", "cheddar", "kapyra biber");  
7 // tam tahıllı ekmekcheddarkapyra biber  
8  
9 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

- ✓ Parametresi olmayan bir fonksiyonu çağırırken istenilen sayıda argüman aktarılabilir. Bunu "arguments nesnesi" (arguments object) özelliğiyle sağlarız. Arguments nesnesi, fonksiyonun parametrelerini dizi olarak tutar. Örneğin

parametresiz olarak oluşturduğumuz "getArguments" isimli fonksiyonumuza 10, 100 ve "Bu arguments nesnesinin 2. dizisi" ifadelerini argüman olarak aktaralım:

```
1 function getArguments() {  
2   console.log(arguments);  
3 }  
4  
5 getArguments(10, 100, "Bu arguments nesnesinin 2. dizisi");  
6 // Arguments(3) [10, 100, "Bu arguments nesnesinin 2. dizisi"]  
7  
8 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

- ✓ Parametre giriyorsak parametrenin veri türünü tanımlamaya gerek yoktur. Bazı programlama dillerinde fonksiyon oluştururken parametrenin veri türünü de girmek zorunludur. Örneğin girilen parametrenin string mi, number mı yoksa başka bir veri türü mü olduğu bildirilmelidir. Ancak JavaScript fonksiyonlarında parametrenin veri türünü tanımlamaya gerek yoktur.
- ✓ Argümanların veri türü denetimi olmadığı için argümanlar istenilen veri türlerinde olabilir.

## 2.9. Fonksiyonun İçinde Fonksiyon Oluşturmak

Bir fonksiyonun içinde bir başka fonksiyon veya fonksiyonlar oluşturulabilir. Ancak temiz ve sürdürülebilir kod yazımı için iç içe fonksiyon yapısı tercih edilmemelidir. Çünkü bir fonksiyon her zaman bir işi yerine getirmelidir.

Örneğin sandviç yapma fonksiyonumuzun içine bir de çay yapma fonksiyonu eklemek çok da iyi olmayacaktır. Çünkü sandviçimizin yanında her zaman çay içmek istemeyebiliriz. Bazen ayran bazen soda bazen başka bir içeceklerle sandviçimizi yemek isteyebiliriz. Ancak sandviç yapma fonksiyonunun içine çay yapma fonksiyonunu eklersek her sandviç yapmaya çalıştığımızda istemediğimiz halde çay yapmak mecburiyetinde kalacağız. Bundan dolayı, çay yapma fonksiyonunu sandviç yapma fonksiyonunun içine koymak hatalı bir yaklaşım olur. Bu iki fonksiyonu iç içe oluşturmak

yerine birbirinden bağımsız oluşturmak daha mantıklı bir yaklaşım olacaktır. Bazı istisna durumlar haricinde iç içe fonksiyon yapısını kullanmamalıyız.

## 2.10. Klasik Anonim Fonksiyon Yapısı

Anonim fonksiyonda fonksiyonun adı yoktur. Anonim fonksiyona isimsiz fonksiyon da denilir.

Anonim fonksiyonun yapısı şu şekildedir:



```
1  (function () {  
2    // function body  
3  })  
4  
5  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Normal fonksiyonda fonksiyonumuzun ismi vardı. Fonksiyonumuzu çağırmak için ismini kullanıyorduk. Ancak anonim fonksiyonun ismi olmadığı için çağıramayacak mıyız? Elbette hayır. Anonim fonksiyonu çağırmanın birkaç yolu var. Birincisi "IIFE" kullanmak. "IIFE" deyince hemen kafanız karışmasın açılımı "Immediately Invoked Function Expression"dır. Bunu, hemen çağırılan fonksiyon ifadesi olarak çevirebiliriz. "();" yapısı IIFE olarak adlandırılır. Bu isimlendirmeyi "Ben Alman" 2010 yılında yapmıştır. Programlama dili ne kadar güzel bir şey değil mi! İnsan dillerinin canlı olup sürekli gelişmesi gibi programlama dilleri de canlı ve sürekli gelişmekte. Kim bilir belki bunu okuyan sen de JavaScript dahil birçok programlama diline katkıda bulunacaksın, hatta bulunmalısın. Çok fazla konuyu dağıtmadan kodlamaya devam edelim! Evet, anonim fonksiyonu çağırmanın birinci yolu fonksiyonu oluşturduktan sonra hemen sonuna parametreyi temsil eden "();" eklemek. Mantıklı değil mi! Çünkü isimli fonksiyonumuzu çağırmak için "functionName();" yapısını kullanıyorduk. Anonim fonksiyonumuzun ismi yoksa o halde "();" yapısını kullanmamız gerekir:



```
1  (function () {  
2    // function body  
3  })();  
4  
5  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Eğer anonim fonksiyonumuzun hemen çalışmasını istemiyorsak bir zamanlayıcı ekleyerek istediğimiz süre sonunda çalışmasını sağlayabiliriz. Fonksiyona zamanlayıcı eklemeyi dokümanımın ilerleyen kısımlarında anlatacağım.

Şimdi, web sayfası açıldığında ekranda “Bu bir anonim fonksiyondur!” yazılı bir popup çıkan bir anonim fonksiyon oluşturalım:



```
1  (function () {  
2    alert('Bu bir anonim fonksiyondur');  
3  })();  
4  
5  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Anonim fonksiyonu çağırmanın birinci yolunun “IIFE” kullanmak olduğunu söyledik, ikinci yol ise anonim fonksiyona bir değişken atayıp fonksiyonumuzu bu değişken içinde tutmak ve sonra bu değişkeni fonksiyonmuş gibi çağırmak. Bu da mantıklı değil mi! Diyelim ki biz normal fonksiyon değil de anonim fonksiyon kullanmanın avantajından yararlanmak istiyoruz. O halde, bir nevi, anonim (isimsiz) fonksiyonumuzu değişken aracılığıyla isimlendirmiş oluruz ve bu fonksiyonu kullanabiliriz. O halde yukarıdaki anonim fonksiyonumuzu IIFE kullanarak değil değişken atayıp bu değişkeni fonksiyonmuş gibi çağırarak oluşturalım:



```
1  const anonFunc = function () {  
2    alert("Bu bir anonim fonksiyondur");  
3  };  
4  
5  anonFunc();  
6  
7  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

### 2.11. Anonim Fonksiyon ve Normal Fonksiyon Arasındaki Farklar

Anonim fonksiyon ve normal fonksiyon arasında birçok fark bulunur. Bunların en önemlilerinden birisi şudur: Normal fonksiyon her yerden çağırılabilirken, değişken atayıp oluşturduğumuz anonim fonksiyon sadece oluşturduğumuz satırın aşağısından çağırılabilir. Örneklerle bunu gösterelim.

Değişken atayıp oluşturduğumuz anonim fonksiyon sadece oluşturduğumuz satırın aşağısından çağırılabilir:



```
1  anonFunc();  
2  /* Bu şekilde hata alırız. Çünkü anonim fonksiyonu  
3     her yerden çağıramayız. Anonim fonksiyonu, değişken  
4     atadığımız satırdan sonra çağırabiliriz. */  
5  
6  const anonFunc = function () {  
7    alert("Bu bir anonim fonksiyondur");  
8  };  
9  
10 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```



Normal fonksiyon ise her yerden çağırılabilir:

```
1  namedFunction();
2  /* Bu şekilde hata almayız. Çünkü adlandırılmış fonksiyonu
3     her yerden çağırabiliriz. */
4
5  function namedFunction() {
6      alert("Bu bir adlandırılmış fonksiyondur");
7  }
8
9  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Anonim fonksiyon ve normal fonksiyon arasındaki en önemli ikinci fark ise şudur: Değişken atanmış anonim fonksiyonumuzda kullandığımız değişkene daha sonra başka bir fonksiyon veya değer atayabiliriz. Bu açıdan anonim fonksiyon bize çok büyük esneklik sağlıyor. Bunları birer örnekle gösterelim.

Değişken atanmış anonim fonksiyonumuzda kullandığımız değişkene daha sonra başka bir fonksiyon atayabiliriz:

```
1  let anonFunc = function () {
2      alert("Bu bir anonim fonksiyondur");
3  };
4
5  // Değişkenimize yeni bir anonim fonksiyon atadık
6  anonFunc = function () {
7      alert("Değişiklik yapıldı");
8  };
9
10 anonFunc();
11 // Değişiklik yapıldı
12
13 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Değişken atanmış anonim fonksiyonumuzda kullandığımız değişkene daha sonra başka bir değer atayabiliriz:

```
1  let anonFunc = function () {
2    alert("Bu bir anonim fonksiyondur");
3  };
4
5  anonFunc = 5;
6  // console.log(anonFunc); // 5
7
8  /* Değişkenimize yeni bir değer atadığımız için
9     artık fonksiyon olarak çağıramayız. */
10
11  anonFunc();
12  // TypeError: anonFunc is not a function
13
14  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

## 2.12. Modern Fonksiyon Yapısı (Arrow Fonksiyon)

JavaScript dili sürekli gelişmekte olan bir dil. Bu gelişimi, ECMAScript versiyonları üzerinden görmek mümkün. Son ECMAScript sürümü haziran 2022'de yayımlanan ES13'tür.


Modern fonksiyon yapısı dediğimiz "arrow fonksiyon" yapısı ise ES6'yla gelmiştir. Arrow'un kelimesinin Türkçesi "ok"tur. Bu fonksiyon yapısına arrow denilmesinin sebebi de fonksiyon yapısında "ok" işaretinin yer almasıdır. Yani bir fonksiyonda "ok (=>)" işareti geçiyorsa bu bir arrow fonksiyondur.

Peki klasik fonksiyon yapısı varken neden modern fonksiyon (arrow fonksiyon) yapısı geliştirilmiştir? Bunun birden çok sebebi var. Performans ve kodun okunabilirliğini kolaylaştırmak için kodları kısaltmak gerekir. Bunun için daha kısa ve kolay yazılan fonksiyon yapısı olarak modern fonksiyon yapısı geliştirilmiştir. Ayrıca modern

## İsa Sarı – Protein & Patika.dev Vue.js Bootcamp Notlarım – 2. Hafta

fonksiyon yapısı, metodların içinde daha iyi çalışır ("this" konusuyla ilgili olan bu duruma bu dokümanın güncel versiyonunda değineceğim).


Arrow fonksiyonu, klasik fonksiyonun yaptığı işi daha az kodla yapar. Arrow fonksiyonun genel yapısı şu şekildedir:



```
1  let functionName = () => {  
2    // code  
3  }  
4  
5  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Yani modern fonksiyon yapısında, fonksiyon bir değişken olarak tanımlanır. Değişkenimizin adı aynı zamanda fonksiyonumuzun adıdır. Normal parantez içine parametreler yazılabilir. Süslü parantez içerisine de kodlarımızı ekleyebiliriz.

Burada fonksiyonumuzu let ile tanımladık. Ancak var veya const ile de tanımlayabiliriz. Hatta let, var, const kullanmadan da tanımlayabiliriz fakat bu kullanım pek önerilmez:



```
1  functionName = () => {  
2    // code  
3  }  
4  
5  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Modern fonksiyon yapısı da klasik fonksiyon yapısı gibi çağırılır:

```
1 functionName();  
2  
3 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Klasik fonksiyon ve modern fonksiyonu kod bloğu açılmış şekilde örneklendirdik, ancak bunları tek satır halinde de yazabiliriz:

```
1 // Klasik fonksiyon yapısı  
2 function functionName() {alert("hello")}  
3  
4 // Modern fonksiyon yapısı  
5 let functionName = () => {alert("hello")}  
6  
7 // Her ikisi de aynı şekilde çağırılır  
8 functionName();  
9  
10 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Arrow fonksiyonun yukarıda gösterdiğimiz genel yapısının dışında farklı yazılışları da vardır:

- ✓ Eğer tek parametre varsa parametrenin parantezini kaldırabiliriz. Ancak birden fazla parametre varsa parametrenin parantezini yazmak zorundayız ve klasik fonksiyonda olduğu gibi parametreleri virgülle ayırmalıyız.



```
1 let introduceYourself = (name) => { return "Benim adım " + name; }
2
3 let introduceYourself = name => { return "Benim adım " + name; }
4
5 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

- ✓ Birden fazla sayıda parametre varsa parametrenin parantezini yazmak zorundayız ve klasik fonksiyonda olduğu gibi parametreleri virgülle ayırmalıyız:



```
1 let introduceYourself = (greeting, name) => { return greeting + ", " + "benim adım " + name; }
2
3 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

- ✓ Eğer işlemimiz tek satırda gerçekleşiyorsa süslü paranteze ve "return" ifadesini kullanmayabiliriz:



```
1 let introduceYourself = name => { return "Benim adım " + name; }
2
3 let introduceYourself = name => "Benim adım " + name;
4
5 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

### 2.13. Fonksiyonun Çalıştırılma Zamanı Nasıl Ayarlanır?

Oluşturduğumuz bir fonksiyonu çağırdığımız anda değil de belirli bir zamandan sonra çalışmasını isteyebiliriz. Bir web sayfasına girdikten 5 saniye sonra çıkan popuplar buna örnektir. Bu popupların ekrana çıkma süresi ayarlanmıştır.

Fonksiyonların çalıştırılma zamanını ertelemek için `setTimeout()` metodu kullanılır. `setTimeout()`'un sözdizimi şu şekildedir:

```
1  setTimeout(fonksiyonumuz, [ertelemesüresi], [argüman1], [argüman2], ...)  
2  
3  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Ancak temiz ve okunabilir kod yazımı açısından `setTimeout()`'a bir değişken atamak gerekir:

```
1  let timerId = setTimeout(fonksiyonumuz, [ertelemesüresi], [argüman1], [argüman2], ...)  
2  
3  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Bir kullanıcı web sayfamızı ziyaret edince "Hoş geldiniz, kampanyamızı gördünüz mü?" şeklinde bir popup çıkan bir fonksiyon oluşturalım. Ancak bu popup 10 saniye sonra çıksın:

```
1  function informCampaign(phrase, campaign) {  
2    alert(phrase + ', ' + campaign);  
3  }  
4  
5  setTimeout(informCampaign, 10000, "Hoş geldiniz", "kampanyamızı gördünüz mü?");  
6  
7  // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

## İsa Sarı – Protein & Patika.dev Vue.js Bootcamp Notlarım – 2. Hafta

Süre, milisaniye cinsinden yazılır. 1 saniye 1000 milisaniyeye eşittir. Bundan dolayı, ben fonksiyonumun çağrıldıktan 10 saniye sonra çalışmasını istediğim için 10000 milisaniye yazdım.

setTimeout() metodunu kullanırken dikkat edilmesi gereken bir durum var. Normalde, fonksiyonumuzu çağırmak için fonksiyonumuzdan sonra ";" yapısını getiririz. Ancak setTimeout içinde bunu yapmamalıyız. Yoksa kodumuz çalışmayacaktır:

```
1 function informCampaign(phrase, campaign) {  
2     alert(phrase + ', ' + campaign);  
3 }  
4  
5 setTimeout(informCampaign(), 10000, "Hoş geldiniz", "kampanyamızı gördünüz mü?");  
6 // Hatalı yazımdan dolayı, hem zamanlama hem de fonksiyon çalışmaz.  
7  
8 setTimeout(informCampaign, 10000, "Hoş geldiniz", "kampanyamızı gördünüz mü?");  
9 // Bu şekilde yazıldığında, hem zamanlama hem de fonksiyon çalışır.  
10  
11 // İsa Sarı - Protein & Patika Vue.js Bootcamp Notlarım
```

Fonksiyonları zamanlama konusunda kullanılan setInterval, clearTimeout, clearInterval gibi metodları bu dokümanın güncellenmiş versiyonunda bulabileceksiniz.

### 2.14. Fonksiyonlar Konusunun Devamı

JavaScript'te fonksiyonlar bu anlattıklarım ile sınırlı değil. Daha detaylı ve ileri seviye konuları, bu dokümanın güncellenmiş versiyonunda bulabileceksiniz. Github hesabımda bu dokümanın güncellenmiş versiyonu yer alacak.

## 3. JavaScript'te Diğer Konular

Bu dokümanın güncellenmiş versiyonunda aşağıdaki konulara da yer vereceğim. Github hesabımda bu dokümanın güncellenmiş versiyonu yer alacak.

- ✓ If Else
- ✓ Array & Object Destructuring
- ✓ Spread & Rest Operator
- ✓ Export & Import
- ✓ JavaScript ile Fetch

## **4. GIT Versiyon Kontrol Sistemi Nedir?**

GIT versiyon kontrol sistemi, bir yazılım projesi veya doküman üzerinde yaptığınız değişiklikleri adım adım izleyen, istediğinizde kaydeden ve isterseniz bunu internet üzerindeki bir bilgisayarda veya yerel bir cihazda saklamanızı ve yönetmenizi sağlayan bir sistemdir. Eğer yazılım projenizi veya bir dokümanınızın versiyonunu yönetmek istiyorsanız bunun için kurulmuş bazı web siteleri var. Bunların en yaygın olanları Github, Gitlab ve BitBucket'tir.

GitHub, işbirliği ve sürüm kontrolü için bir kod barındırma platformudur. GitHub, sizin ve çalışma arkadaşlarınızın projeler üzerinde birlikte çalışmanıza olanak tanır.

Github'ı kullanmak için başlangıç seviye bilgilerine şu linkten erişebilirsiniz: <https://github.com/skills/introduction-to-github>

Bu dokümanımın güncellenmiş versiyonunda Github'la ilgili detaylı bilgiler bulabileceksiniz. Bu dokümanımın güncellenmiş versiyonuna Github adresimden erişebilirsiniz.

## **5. Chrome DevTools nedir?**

Chrome DevTools, web geliştirme aracıdır. DevTools, sayfaları anında düzenlemenize ve sorunları hızlı bir şekilde teşhis etmenize yardımcı olabilir, bu da sonuçta daha iyi web sitelerini daha hızlı oluşturmaya yardımcı olur.

Visual Studio Code'da kodlama yaparken Live Server'dan anında çıktı alabiliriz. Eğer tarayıcı olarak Chrome kullanırsak kodlarımızda hata meydana geldiğinde bunu hızlıca çözmek için Chrome DevTools'u kullanabiliriz. Bu araçla sadece hata yönetimi değil, web arayüzündeki elementlere erişmek, ağ analizi ve takibi, hafıza kontrolü ve daha birçok şey için kullanabiliriz.

DevTools'ta Elements, Console, Sources, Network, Performance, Memory ve daha birçok sekme sayesinde birçok işlem gerçekleştirmek mümkün.

DevTools'a web sayfası üzerinde sağ tıklayıp Inspect (İncele) ile erişilebilir.

Not: Chrome DevTools, Chrome tarayıcısında yerleşik olarak bulunmaktadır, kurulum gerekmemektedir.

Daha detaylı bilgi için bakınız: <https://developer.chrome.com/docs/devtools/open/>



## **6. ViteJS nedir?**

Vite.js, hızlı ve basit bir web geliştirme aracıdır. Vite'i geliştiren kişi aynı zamanda Vue'yu geliştiren Evan You'dur.

Vite kendi içerisinde compiler kullanıyor. Yazmış olduğumuz ECMAScript (ES) versiyonlarını, tarayıcıların anlayabileceği şekle çeviriyor.

JavaScript sürekli gelişen bir dil. Bu gelişmeleri, farklı zamanlarda yayınlanan ES versiyonları ile görebiliyoruz. Günümüze kadar birçok ES Modülü yayınlanmıştır. Bunlar ES 5, ES 6, ES 7, ES 8, ES 9, ES 10, ES 11, ES 12 ve son olarak haziran 2022'de yayınlanan ES 13.

Tarayıcılar, tüm ES modüllerini desteklemiyor olabilir. Örneğin, ES 11'de yayınlanan bir özelliği kullandığımızda tarayıcı bu özelliği algılamayabilir. Bundan dolayı, Vite gibi araçlara ihtiyaç duyarız. Vite'den önce, yaygın olarak kullanılan webpack, Rollup ve Parcel gibi araçlar bulunmaktaydı. Vite ise rakiplerinden daha iyi olduğunu belirtiyor.

Eğer yeni bir projeye başlıyorsanız Vite kullanmanızı öneririm.

Vite'in neden tercih edilmesiyle ilgili detaylı bilgiye şuradan ulaşabilirsiniz: <https://vitejs.dev/guide/why.html>

Vite'in kurulumu için bakınız: <https://vitejs.dev/guide/>

ES Modülleriyle ilgili bazı bilgilere şuradan ulaşabilirsiniz: <https://en.wikipedia.org/wiki/ECMAScript>

Not: Vite'i sadece Vue projelerinizde değil aynı zamanda React, Angular ve Svelte projelerinizde de kullanabilirsiniz. Ayrıca pure JavaScript yazarken de kullanabilirsiniz.

## **7. Vue.js'e Giriş**

Bootcamp'in 2. Haftasında Vue.js'e giriş yaptık. Bu notlarımı, bu dokümana dahil etmedim. Yayımlayacağım 3. Hafta notlarımda bulabileceksiniz.

## **8. Yararlı Linkler / Kaynaklar**

Bootcamp'ın 2. Haftasında eğitmenin, bootcamp katılımcılarının ve asistanların paylaştığı yararlı linkler ile bu dokümanı oluştururken yararlandığım bazı kaynakların listesi:

- 1) <https://javascript.info/arrow-functions>
- 2) <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions>
- 3) [https://www.w3schools.com/js/js\\_arrow\\_function.asp](https://www.w3schools.com/js/js_arrow_function.asp)
- 4) [https://www.w3schools.com/js/js\\_functions.asp](https://www.w3schools.com/js/js_functions.asp)
- 5) <https://app.patika.dev/courses/javascript>
- 6) <https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/>
- 7) <https://www.freecodecamp.org/news/javascript-map-reduce-and-filter-explained-with-examples/>
- 8) [https://github.com/Asabeneh/30-Days-Of-JavaScript/blob/master/05 Day Arrays/05 day arrays.md#methods-to-manipulate-array](https://github.com/Asabeneh/30-Days-Of-JavaScript/blob/master/05%20Day%20Arrays/05_day_arrays.md#methods-to-manipulate-array)
- 9) <https://codeburst.io/map-filter-and-reduce-in-javascript-728f2b9ace8?gi=bf89c4065f03>
- 10) [https://developer.mozilla.org/en-US/docs/Web/API/Document Object Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)
- 11) <https://github.com/public-apis/public-apis>
- 12) [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional chaining](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining)
- 13) <https://www.atlassian.com/software/jira>
- 14) <https://ohmyz.sh/>
- 15) <https://hyper.is/>
- 16) <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
- 17) <https://app.patika.dev/courses/git/git-versiyon-kontrol-sistemi-nedir>
- 18) JavaScript, Axel Rauschmayer, 2021.