Swift Programlama Dili ile iOS Mobil Uygulama Geliştirme

MURAT ADIGÜZEL (murat.adiguzel@intertech.com.tr)

5 Nisan 2021

Fonksiyonlar

Fonksiyonlar

Egzersizler

1. `firstName` ve `lastName` adlı iki `String` alan `printFullName` adlı bir fonksiyon yazın. Fonksiyon, `firstName` + `" "` + `lastName` olarak tanımlanan tam adı yazdırmalıdır. Kendi tam adınızı yazdırmak için kullanın.

2. `printFullName` tanımını, her iki parametre için de harici herhangi bir ada sahip olmayacak şekilde değiştirin.

3. Tam adı bir `String` olarak döndüren `calculateFullName` adlı bir fonksiyon yazın. Kendi tam adınızı sabit olarak saklamak için kullanın.

4. `calculateFullName` fonksiyonunu, hem tam adı hem de adın uzunluğunu içeren bir **`tuple`** döndürecek şekilde değiştirin. `count` özelliğini kullanarak bir `String`'in uzunluğunu bulabilirsiniz. Kendi tam adınızın uzunluğunu bulmak için bu fonksiyonu kullanın.

Fonksiyonlar

Egzersizler

1. Son bölümde, sayılabilir aralıklara sahip **`for`** döngüleri yazdınız. Sayılabilir aralıklar, her zaman birer birer artmaları gerektiğinden sınırlıdır. **`stride(from:to:by:)`** ve **`stride(from:through:by:)`** fonksiyonları çok daha esnek bir şekilde döngü yapmanıza olanak tanır.

   Örneğin, `10`'dan `20`'ye `4`'er atlayarak bir döngü yapmak istiyorsanız şunları yazabilirsiniz:

```
for index in stride(from: 10, to: 22, by: 4) {
  print(index)
}
for index in stride(from: 10, through: 22, by: 4) {
  print(index)
}
```

   - Aşırı yüklenen iki stride fonksiyonunun arasındaki fark nedir?

   - `10.0`'dan `9.0`'a (`9.0` dahil) `0.1` azalan bir döngü yazın.

Fonksiyonlar

Egzersizler

2. Asal sayılar, sadece kendisine ve 1 sayısına kalansız bölünebilen 1'den büyük pozitif tam sayılardır. İlk olarak aşağıdaki imzaya sahip bir fonksiyon yazın:

   ```
   func isNumberDivisible(_ number: Int, by divisor: Int) -> Bool
   ```

   Bu fonksiyonu bir sayının başka bir sayıya tam bölünüp bölünmediğine karar vermek için kullanacaksınız. Burada number değeri, divisor değerine tam bölünüyorsa fonksiyon **`true`**, aksi takdirde **`false`** dönmelidir.

   İpucu: Fonksiyonda modulo **`(%)`** operatörünü kullanabilirsiniz.

   Daha sonra aşağıdaki imzaya sahip olan ana fonksiyonu yazın:

   ```
   func isPrime(_ number: Int) -> Bool
   ```

   Bu fonksiyon, sayı asalsa **`true`**, aksi takdirde **`false`** döndürmelidir. **`1`**'den **`number`**'a kadar olan sayıları gözden geçirmeli ve sayının bölenlerini bulmalısınız. 1 ve kendisinden başka bölenleri varsa, sayı asal değildir. Burada ilk yazdığınız **`isNumberDivisible`** fonksiyonunu kullanmalısınız.

   Fonksiyonu aşağıdaki durumları test etmek için kullanın:

   ```
   isPrime(6) // false

   isPrime(13) // true

   isPrime(8893) // true
   ```

   İpucu 1: `0`'dan küçük sayılar asal sayılmamalıdır. Fonksiyonun başlangıcında bu durumu kontrol edin ve sayı `0`'dan küçükse erken dönün.

   İpucu 2: Bölenleri bulmak için bir for döngüsü kullanın. 2'den başlayıp, sayının kendisinden önce biterseniz, bir bölen bulur bulmaz, false döndürebilirsiniz.

   İpucu 3: Fonksiyonu daha da optimize etmek için sayının kendisine kadar gitmek yerine, 2'den sayının karekköküne ulaşana kadar bir döngü yapabilirsiniz.

Fonksiyonlar

Egzersizler

3. Bu egzersizde, bir fonksiyonun kendini çağırdığında ne olacağını göreceksiniz. Bu davranışa **recursion** adı verilir.

   Fibonacci dizisinden bir değer hesaplayan bir fonksiyon yazacaksınız. Dizideki herhangi bir değer, önceki iki değerin toplamıdır. Dizi, ilk iki değer 1'e eşit olacak şekilde tanımlanmıştır. Yani, **fibonacci(1) = 1** ve **fibonacci(2) = 1**'dir.

   Aşağıdaki imzayı kullanarak fonksiyonu yazın:

   ```
   func fibonacci(_ number: Int) -> Int
   ```

   Ardından, aşağıdaki sayılar ile çalıştırarak fonksiyonu doğru yazdığınızı doğrulayın:

   ```
   fibonacci(1)  // = 1
   fibonacci(2)  // = 1
   fibonacci(3)  // = 2
   fibonacci(4)  // = 3
   fibonacci(5)  // = 5
   fibonacci(10) // = 55
   ```

   İpucu 1: `0` ve daha küçük sayılar için `0` döndürmelisiniz.

   İpucu 2: Diziyi başlatmak için, sayı 1 veya 2'ye eşit olduğunda 1 dönüş değerini sabit kodlayın.

   İpucu 3: Diğer herhangi bir değer için, fibonacci fonksiyonunu **`number - 1`** ve **`number - 2`** değerleri ile çağırıp toplamını döndürmeniz gerekir.

Fonksiyonlar

Özet

1. Kodu birden çok kez yazmak zorunda kalmadan istediğiniz kadar çalıştırabileceğiniz bir görevi tanımlamak için bir fonksiyon kullanırsınız.

2. Fonksiyonlar sıfır veya daha fazla parametre alabilir ve isteğe bağlı olarak bir değer döndürebilir.

3. Bir fonksiyon çağrısında kullandığınız etiketi değiştirmek için bir fonksiyon parametresine harici bir ad ekleyebilir veya etiket olmadığını belirtmek için bir alt çizgi (wildcard operator) kullanabilirsiniz.

4. Parametreler, siz onları `inout` (copy-in copy-out) olarak işaretlemedikçe sabit olarak aktarılır.

5. Fonksiyonlar, farklı parametrelerle aynı ada sahip olabilir. Buna **aşırı yükleme (overloading)** denir.

Fonksiyonlar

Özet

6. Fonksiyonlar, Swift'e bu fonksiyonun asla dönmeyeceğini bildirmek için özel bir `Never` geri dönme türüne sahip olabilir.

7. Değişkenlere fonksiyonlar atayabilir ve bunları diğer fonksiyonlara aktarabilirsiniz.

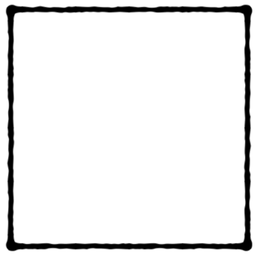
8. Düzgün şekilde adlandırılmış ve tek bir işi olan fonksiyonlar oluşturmaya çalışın.

9. Fonksiyon dokümantasyonu, fonksiyonun önüne `///` ile bir açıklama bölümü eklenerek oluşturulabilir.

Optional Türler

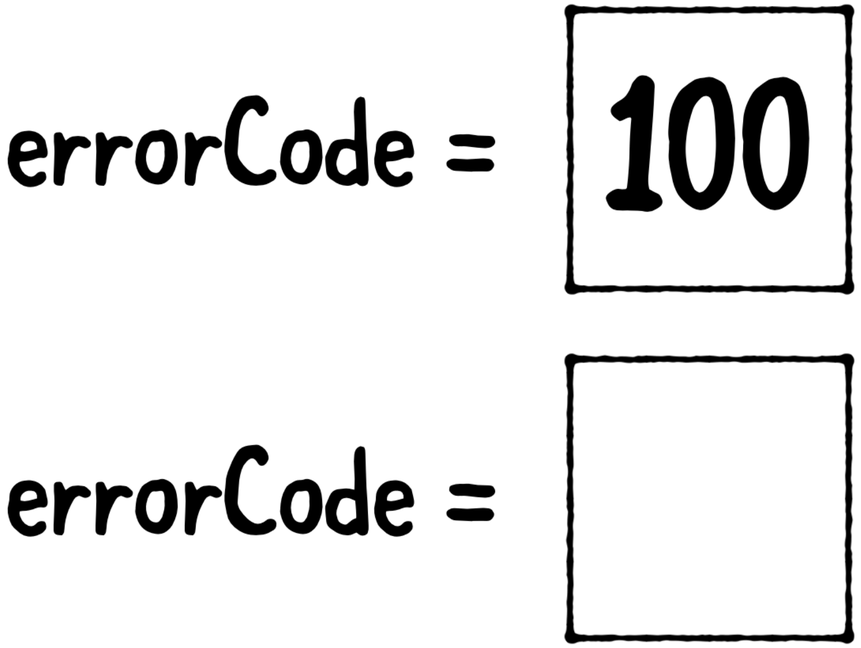
Optional Türler

Optional box
containing a value



Optional box
containing no value

Optional Türler



Optional Türler

Egzersizler

1. `myFavoriteSong` adında **Optional** bir `String` oluşturun. Favori bir şarkınız varsa, onu o şarkıyı ifade eden bir `String`'e ayarlayın. Birden fazla favori şarkınız varsa veya hiç favoriniz yoksa, **Optional**'ı `nil`'e ayarlayın.

2. `parsedInt` adında bir sabit oluşturun ve onu `"10"` `String` değerini bir `Int` değere dönüştürmeye çalışan `Int("10")` değerine eşit olarak ayarlayın. Option tıklamayı kullanarak dönüştürülmüş `Int` türünü kontrol edin. Neden **Optional**'dır?

3. Yukarıdaki egzersizde dönüştürülen `String` değeri sayı olmayan bir `String` ile değiştirin (Örneğin `"Dog"` değerini deneyin). `parsedInt` şimdi neye eşit?

Optional Türler

Egzersizler

1. Önceki `myFavoriteSong` değişkeninizi kullanarak, bir değer içerip içermediğini kontrol etmek için **optional binding** kullanın. Varsa, değeri yazdırın, yoksa `"I don't have a favourite song."` yazısını yazdırın.

2. `myFavoriteSong`'u şimdi olduğu şeyin tersine değiştirin. `nil` ise, bir `String` değere ayarlayın; eğer bir `String` ise `nil` olarak ayarlayın. Sonucunuzun nasıl değiştiğini gözlemleyin.

Optional Türler

Egzersizler

1. Hangi ifadeler geçerlidir?

```
var name: String? = "Murat"

var age: Int = nil

let distance: Float = 26.7

var middleName: String? = nil
```

Optional Türler

Egzersizler

2. İlk olarak, bir tamsayının başka bir tam sayıya kalansız kaç kez bölünebileceğini döndüren bir fonksiyon yazın. Bölme işlemi bir tam sayı üretmezse fonksiyon `nil` döndürmelidir. Fonksiyonu **`divideIfWhole`** olarak adlandırın.

   Ardından, fonksiyonun **Optional** sonucunu çözmeye çalışan bir kod yazın. İki durum olması gerekir: Başarı durumunda `"Yep, it divides \(answer) times"`, başarısız olduğunda `"Not divisible :["` yazdırın.

   Son olarak fonksiyonunuzu test edin:

   `10` ve `2` değerleri için `"Yep, it divides 5 times"` yazmalıdır.

   `10` ve `3` değerleri için `"Not divisible :["` yazmalıdır.

   İpucu 1: Aşağıdakileri fonksiyon imzasını kullanın:

   ```
   func divideIfWhole(_ value: Int, by divisor: Int)
   ```

   Optional bir dönüş türü eklemeniz gerekiyor.

   İpucu 2: Bir değerin diğeriyle bölünebilir olup olmadığını belirlemek için modulo operatörünü `(%)` kullanabilirsiniz; bu işlemin iki sayının bölünmesinden kalanı döndürdüğünü hatırlayın. Örneğin, `10 % 2 = 0`, **`10`**'un **`2`**'ye kalansız bölünebileceği anlamına gelirken, `10 % 3 = 1`, **`10`**'un **`3`**'e **`1`** kalanı ile bölünebileceği anlamına gelir.

Optional Türler

Egzersizler

3. Son egzersizdeki test kodları `if` ifadesi kullanılarak yazıldı. Yazılan bu kodları **nil-coalescing** kullanarak yeniden düzenleyin. Bu kez her durumda `"It divides X times"` yazdırmasını sağlayın, ancak bölme bir tam sayı ile sonuçlanmazsa, X'in `0` olması gerekir.

4. Aşağıdaki iç içe **optional** tanımını kontrol edin. Bunu iç içe yerleştirilmiş üç kutu içinde bir sayı olarak düşünebilirsiniz.

```
let number: Int??? = 10
```

`number` sabitini yazdırdığınızda aşağıdaki sonucu alırsınız:

```
print(number)
// Optional(Optional(Optional(10)))
print(number!)
// Optional(Optional(10))
```

Şunları yapın:

1. Tamamen **force unwrapping** kullanın ve sayıyı yazdırın.
2. **Optional binding** kullanarak sayıyı yazdırın.
3. Yalnızca çözülebiliyorsa sayıyı yazdırmak için `guard` kullanan `printNumber(_ number: Int???)` fonksiyonunu yazın.

Optional Türler

Özet

1. `nil`, bir değerin yokluğunu ifade eder.

2. **Optional** olmayan değişkenler ve sabitler asla `nil` olamazlar.

3. **Optional** değişkenler ve sabitler, bir değer içerebilen veya `nil` olabilen kutular gibidir.

4. **Optional** bir değerin içindeki değerle çalışmak için, önce **Optional**'ı çözmelisiniz.

5. **Optional** bir değeri çözmenin en güvenli yolu, **optional binding** veya **nil-coalescing** kullanmaktır. **Force unwrapping** yöntemini yalnızca uygun olduğunda kullanın, çünkü bu bir çalışma zamanı hatasına neden olabilir.

6. **Optional binding** için `guard let` kullanabilirsiniz. **Binding** başarısız olursa, derleyici sizi fonksiyondan çıkmaya zorlar. Bu, programınızın hiçbir zaman ilklendirilmemiş bir değerle çalışmayacağını garanti eder.