Koşul, Döngü ve Fonksiyonlar

Dinçer GÖKSÜLÜK, Phd.

13 Mayıs 2022

Contents

Gereksinimler:

- **R 4.2.0** sürümü.
- **RStudio Desktop 2022.02.2+485** veya üzeri sürümler.
- MAC Kullanıcılar için **XQuartz** güncel sürümü.

Koşul ve Döngüler

if ... else ...

- Fonksyionlar yazımı ve veri analizlerinde en çok kullanılan yapılardan birisi mantıksal sınamalardır.
- Verilen koşulun sağlandığı veya sağlanmadığı durumlarda dikkate alınması gereken kodlamalar için kullanılır.
- İki durumlu veya ikiden falza durumlu koşullar için mantıksal sınamalar yazılabilir.
- İki durumlu mantıksal sınamalar için kullanılan bir diğer fonksiyon ise `ifelse(<koşul>, <doğru ise>, <yanlış ise>)` fonksiyonudur.

```
# İki durumlu mantıksal sınama:
if (koşul){
  <Koşul doğru ise>
} else {
  <Koşul yanlış ise>
}

## İkiden fazla durumlu mantıksal sınama:
if (koşul1){
  <Koşul1 doğru ise>
} else if (koşul2){
  <Koşul2 yanlış ise>
} ... {

} else {
```

```
  <Yukarıdaki koşulların hiç birisi doğru değil ise>
}
```

```
Cinsiyet <- "K"

if (Cinsiyet == "K"){
  Cinsiyet <- "Kadın"
  print("K olarak girilmiş olan cinsiyet bilgisi 'Kadın' olarak yeniden düzenlendi.")
} else {
  Cinsiyet <- "Erkek"
  print("E olarak girilmiş olan cinsiyet bilgisi 'Erkek' olarak yeniden düzenlendi.")
}
```

```
## [1] "K olarak girilmiş olan cinsiyet bilgisi 'Kadın' olarak yeniden düzenlendi."
```

Yukarıda kullanılan `if .. else ...` mantıksal sınamasını `ifelse(<koşul>, <doğru ise>, <yanlış ise>)` yapısı kullanarak da düzenlemek mümkündür.

```
cinsiyet <- "K"
cinsiyet2 <- ifelse(cinsiyet == "K", "Kadın", "Erkek")

print(cinsiyet2)
```

Örnek: Kadınlar ve erkeklerde yüksek kolesterol düzeyleri sırasıyla 180 mg/dL ve 200 mg/dL olarak dikkate alınsın. Aşağıda bilgileri verilen kişilerin yüksek kolesterol tanısı alıp almadığını belirleyelim.

Ad	Soyad	Kolesterol	Cinsiyet
Dinçer	Göksülük	182	1
Ayşe	Aslan	194	2

```
# Kişilere ait bilgiler bir listede birleştirilmiştir.
person1 <- list(Ad = "Dinçer", Soyad = "Göksülük", Kolesterol = 182, Cinsiyet = 1, Tanı = NA)
person2 <- list(Ad = "Ayşe", Soyad = "Aslan", Kolesterol = 194, Cinsiyet = 2, Tanı = NA)

if (person1$Cinsiyet == 1){
  if (person1$Kolesterol >= 200 ){
    tani1 <- "Pozitif"
  } else {
    tani1 <- "Negatif"
  }
} else {
  if (person1$Kolesterol >= 180 ){
    tani1 <- "Pozitif"
  } else {
    tani1 <- "Negatif"
  }
}

person1$Tanı <- tani1

if (person2$Cinsiyet == 1){
  if (person2$Kolesterol >= 200 ){
    tani2 <- "Pozitif"
  } else {
```

```
    tani2 <- "Negatif"
  }
} else {
  if (person2$Kolesterol >= 180 ){
    tani2 <- "Pozitif"
  } else {
    tani2 <- "Negatif"
  }
}

person2$Tanı <- tani2

data <- list(person1, person2)
data
```

```
## [[1]]
## [[1]]$Ad
## [1] "Dinçer"
##
## [[1]]$Soyad
## [1] "Göksülük"
##
## [[1]]$Kolesterol
## [1] 182
##
## [[1]]$Cinsiyet
## [1] 1
##
## [[1]]$Tanı
## [1] "Negatif"
##
##
## [[2]]
## [[2]]$Ad
## [1] "Ayşe"
##
## [[2]]$Soyad
## [1] "Aslan"
##
## [[2]]$Kolesterol
## [1] 194
##
## [[2]]$Cinsiyet
## [1] 2
##
## [[2]]$Tanı
## [1] "Pozitif"
```

Gerekli görüldüğü durumda iç içe `if ... else ...` mantıksal sınamaları oluşturulabilir. Ancak, oluşturulan katman sayısı arttıkça hesaplama sürelerinin artacağı unutulmamalıdır.

- If mantıksal sınaması içerisinde kullanılan koşul değişkeni vektör olduğu durumda koşulun sağlanıp sağlanmadığı dikkatle incelenmelidir.
- If koşulu vektör olarak verilmiş ise vektörün ilk elemanının TRUE/FALSE bilgisi dikkate alınır diğer

elemanlar dikkate alınmaz. **Bu durum R 4.2.0 sürümü itibari ile hata mesajı verecek şekilde düzenlenmiştir.** R 4.2.0 öncesi sürümlerde yapılan analizlerde `if (...)` sınamasında kullanılan koşulun vektör olması durumunda yalnızca ilk eleman kullanılır ve buna yönelik aşağıdaki uyarı mesajı döndürülür.

Warning: the condition has length 1 and only the first element will be used.

```
cinsiyet <- c(1, 2, 1, 2, 2)  # 1: Erkek  2: Kadın
cinsiyet == 1

# R 4.2.0 sürümü itibari ile hata mesajı olarak dönmektedir.
# 4.2.0 öncesi sürümlerde bu durum uyarı mesajı olarak dönecektir.
if (c(cinsiyet == 1)){
  cat("\n Cinsiyet: Erkek")
} else {
  cat("\n Cinsiyet: Kadın")
}
```

Örnek: [1, 3, 4, 5, 11] vektörü içerisinde 10'dan büyük bir değer olup olmadığını belirleyelim.

```
x <- c(1, 3, 4, 5, 11)

x > 10

if (x > 10){
  cat("\nVeri setinde 10'dan büyük değer vardır.")
} else {
  cat("\nVeri setinde 10'dan büyük değer yoktur.")
}
```

Bir vektör üzerinde yapılan mantıksal sınamalarda iki durum araştırılabilir.

- Vektör içerisinde en az 1 TRUE veya FALSE sonucun olup olmadığı (`any(...)` fonksiyonu)
- Vektör içerisindeki sonuçların tamamının TRUE veya FALSE olup olmadığı (`all(...)` fonksiyonu)

```
# x vektöründe 10'dan küçük değerin olup olmadığının araştırılması
sonuc <- (x < 10)  # TRUE/FALSE vektörü

# sonuc vektöründe en az 1 TRUE olup olmadığı
any(x < 10)

# sonuc vektöründe bütün değerlerin TRUE olup olmadığı
all(x < 10)
```

İpucu: TRUE/FALSE türünden bir vektör `as.numeric(...)` fonksiyonu ile **numeric** sınıfında bir vektöre dönüştürülebilir. Bu durumda TRUE değerler 1, FALSE değerler 0 olarak dönüştürülür.

```
# x vektöründe 10'dan küçük değerin olup olmadığının araştırılması
sonuc <- (x < 10)  # TRUE/FALSE vektörü

# TRUE/FALSE vektörünün 1/0 vektörüne dönüştürülmesi
sonuc2 <- as.numeric(sonuc)

# sonuc2 vektörü yardımı ile x vektöründe 10'dan küçük değer olup olmadığını
# nasıl belirleyebiliriz?
```


Çoklu koşul tanımlama

- Aynı anda iki veya daha fazla koşul birlikte değerlendirilmek istendiği durumda her bir koşul **VE**, **VEYA** kuralına göre birleştirilebilir.
- **VE** koşulu için `&`, veya koşulu için ise `|` kullanılır.

Örnek: Yüksek kolesterol tanısı için oluşturulan örnek veriyi kullanarak **person1** ve **person2**'nin *pozitif tanı alan erkek hasta* olup olmadığını araştıralım.

```
person1 <- list(Ad = "Dinçer", Soyad = "Göksülük", Kolesterol = 182, Cinsiyet = 1, Tanı = "Negatif")
person2 <- list(Ad = "Ayşe", Soyad = "Aslan", Kolesterol = 194, Cinsiyet = 2, Tanı = "Pozitif")

if (person1$Cinsiyet == 1 & person1$Tanı == "Pozitif"){
  cat("Person1 pozitif tanı alan bir erkek hastadır.")
} else {
  cat("Person1 pozitif tanı alan bir erkek hasta değildir.")
}

if (person2$Cinsiyet == 1 & person2$Tanı == "Pozitif"){
  cat("Person2 pozitif tanı alan bir erkek hastadır.")
} else {
  cat("Person2 pozitif tanı alan bir erkek hasta değildir.")
}
```

Koşul Sınamalarında `any(...)`, `all(....)`, `&&` ve `||` Kullanımı

- `&` ve `|` operatörleri vektör düzeyinde koşul sınamaları yaparken yine vektör düzeyinde sonuçlar döndürür.
- `&&` ve `||` operatörleri vektörlerin yalnızca ilk elemanlarını dikkate alarak koşul sınaması yapar.

```
v1 <- c(TRUE, FALSE, TRUE, TRUE)
v2 <- c(TRUE, TRUE)
v3 <- c(FALSE, FALSE, TRUE)

v1 & v2
v2 & v1
v1 | v2

v1 & v3  # Uyarı veriyor. Neden?

# R 4.2.0 sürümünde uyarı veriyor.
# 4.2.1 ve sonraki sürümlerde && ve || kullanımlarında da logical vektör kullanılması durumunda hata me
v1 && v3
```

Koşulların `( )` ile Gruplandırılması

- Koşullar arası öncelik ve/veya birliktelik sıralamasını ayarlamak için belirli koşul grupları `( )` ile gruplandırılır.
- Gruplandırılmış koşullar arası sınamalar diğer sınamalara benzer şekilde yapılır.
- **Örnek**: Girilen iki sayısal değer (x1 ve x2) için bu sayılardan herhangi birisinin 20'den küçük bir sayı **YA DA** 40'dan büyük çift sayı olup olmadığını kontrol ediniz.

```
x1 <- 40
x2 <- 17

# x1 ve x2 sayılarının 20'den küçük veya  50'dan büyük bir çift sayı olup
# olmadığını belirleyelim.
```



```
# Çift sayı:
kalan <- x1 %% 2  # x1'in 2'ye bölümünden kalan

# if ((x1 20'den küçük çift sayı) VEYA (x1 40'dan büyük çift sayı)) {
#   ....
# } else {
#   ....
# }
```

for döngüsü

- **for** döngüleri bir dizinin elemanlarını sırayla dikkate alarak istenilen işlemleri yapar.
- İşlemler birisi bittiği anda diğerine geçilecek şekilde dikkate alındığı için büyük çaplı döngülerde işlem zamanında ciddi artmalar olabilir.
- **for** döngülerinin paralel olarak kodlanması işlem zamanında ciddi azalma sağlayabilir. Bu konuda detaylı bilgi için **Parallel Computing**, **foreach**, **doParallel** konularını araştırabilirsiniz.

```
for (<dizi>){
  <komutlar>
}
```

Örnek: [1, 100] aralığında 7'ye bölünen rakamların seçilmesi

```
x <- 1:100
# x <- seq(from = 1, to = 100, by = 1)

## Yöntem 1:
sonuc <- NULL
for (i in 1:length(x)){
  if (x[i] %% 7 == 0){
    sonuc <- c(sonuc, x[i])
  }
}

## Yöntem 2:
sonuc2 <- NULL
for (i in x){
  if (i %% 7 == 0){
    sonuc2 <- c(sonuc2, i)
  }
}

rbind(sonuc, sonuc2)
```

```
##        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## sonuc     7   14   21   28   35   42   49   56   63    70    77    84    91
## sonuc2    7   14   21   28   35   42   49   56   63    70    77    84    91
##        [,14]
## sonuc     98
## sonuc2    98
```

> **Dikkat**: For döngülerinde her adımda elde edilen sonuçların `c(...)`, `cbind(...)`, `rbind(...)` gibi birbiri ardına eklemeli yapılar olarak işlenmesi hesaplama süresini ciddi derecede arttırmaktadır.


```r
# 1 ile 100,000 arasındaki bütün değerlerin logaritmasını üç farklı yöntem ile
# hesaplayalım.

x <- 1:100000
# x <- seq(from = 1, to = 100, by = 1)

# Yöntem 1: Vektörel hesaplama
# log(...) fonksiyonu içerisinde vektör kullanılır ise logaritma işlemi vektörün
# her elemanına hızlı bir şekile uygulanır.
system.time({
  logx1 <- log(x)
})

# Yöntem 2: for döngüsü ile birlikte elde edilen her sonucun birbiri ardına
# eklenerek bütün değerlerin logaritmasının alınması.
system.time({
  logx2 <- NULL
  for (i in 1:length(x)){
    logx2 <- c(logx2, log(x[i]))
  }
})
```

ÇÖZÜM: For döngüsü içerisinde her adımda elde edilen sonuçların daha önceden hazırlanmış bir sonuç vektörü içerisine yazdırılması ile hesaplama süresi ciddi anlamda azaltılabilecektir.

```r
# x vektörü ile aynı boyutta bütün elemanları 0 olan bir sonuç vektörü
# oluşturulur. Her adımda elde edilen sonuç, ilgili vektörün ilgili elemanına
# aktarılır.

logx3 <- numeric(length(x))
system.time({
  for (i in 1:length(x)){
    logx3[i] <- log(x[i])
  }
})

# Elde edilen sonuçların aynı olup olmadığının kontrol edilmesi
identical(logx1, logx2, logx3)
```

Soru: [1, 100] arasındaki tam sayıların ortalamasını `for` döngüsü kullanarak hesaplayınız.

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad i \in \mathbb{R}$$

```r
x <- 1:100

for (<dizi>){
  <komutlar>
}

# x vektörünün ortalaması
```

> **for** döngülerinin süre sorununu aşmak için kullanılabilecek bir yöntem vektör tabanlı kodlamalar (vectorization) kullanmaktır. Bu amaçla `apply(...)`, `lapply(...)`, `sapply(...)`, vb. fonksiyonlar kullanılır.

while döngüsü! Sonsuz bir döngünün içinde olabilirsiniz...

- While döngüsü dikkatle kulanılması gereken bir döngüdür.
- **Convergence** problemlerinde tercih edilebilecek bir döngüdür.
- Koşul bozulmadığı sürece döngü içi hesaplamalar devam eder.
- Ya koşul hiç bozulmaz ise?? **Sonsuz döngü....**

```
while (koşul){
  <kodlar>
}
```

Soru: [1, 100] arasındaki tam sayıların ortalamasını `while` döngüsü kullanarak hesaplayalım.

```
x <- 1:100
n <- length(x)

i <- 1
toplam <- 0

# while döngüsüne girebilmek için koşulun sağlanmış olması gerekir.
# Aksi durumda döngü çalıştırılmadan bir sonraki kod satırlarına geçilir.
while (i < n){
  toplam <- toplam + x[i]
  i <- i + 1   ## Koşulun incelendiği eleman döngü içinde
               ## düzenli olarak güncellenmelidir.
}
mu <- toplam / n

print(mu)
```

> **while** döngüsü koşula bağlı çalıştığı için her adımda koşulun sağlanıp sağlanmadığını kontrol eder. Bu nedenle **for** döngüsüne kıyasla daha yavaştır.

Soru: Bir bölgede Spinal Müsküleratrofi (SMA) görülme sıklığını araştırmak isteyen bir araştımacı bölgedeki bütün kliniklere gelen hastaların dosyalarını incelemiştir. SMA'nın görülme sıklığının 10 binde 1 olduğu bilindiğine göre ilk SMA vakası görülünceye kadar incelenmesi gereken dosya sayısını belirlemek için bir `while` döngüsü oluşturunuz. İlk vakayı kaçıncı gözlemde bulduğunuzu belirleyiniz.

EK Bilgi 1: Incelenen her bir dosyanın SMA vakası içeren bir dosya olup olmadığına aşağıdaki kod ile karar verebilirsiniz. Buradan üretilen sayı 0 ise SMA negatif, 1 ise SMA pozitif olarak değerlendiriniz.

```
# 10 binde 1 ihtimal ile Bernoulli dağılımından sonuç 1 veya 0 olacak
# şekilde veri üretiliyor.
sma_sonuc <- rbinom(n = 1, size = 1, prob = 1/10000)
sma_sonuc
```

apply Fonksiyonları

- Belirli bir işlemin/fonksiyonun bir obje grubuna (list, data.frame, matrix, vector, vb.) sıra ile uygulanması amacı ile geliştirilmiş fonksiyon ailesidir.
- Vektörel işlem yaptığı için **for** döngüsüne kıyasla daha hızlıdır. Güncel R sürümlerinde **for** döngüsü performansı geliştirilmiş olmasına karşın halen `apply` fonksiyon ailesine göre daha yavaştır.
- `apply`, `lapply`, `sapply`, `tapply`, `vapply`, `mapply`, ... gibi farklı türevleri vardır. Her fonksiyonun farklı objeler için farklı çalışma prensibi vardır.
- **for** döngüsünün aksine her işlem birbirinden bağımsız olarak gerçekleşir. Dolayısıyla, bir adımda elde edilen sonuç bir sonraki adımda kullanılamaz.

- `sapply(...)` fonksiyonu

Örnek: [1, 2, 3, 4, 5] vektörünün her bir elemanının karesini alan fonksiyonu `apply` fonksiyonlarını kullanarak yazalım.

```
x <- 1:5

# Yöntem 1:
x^2

# Yöntem 2:
n <- length(x)   # x vektörünün eleman sayısı
x_kare <- numeric(n)

for (i in 1:n){
  x_kare.i <- x[i] * x[i]
  x_kare[i] <- x_kare.i
}
print(x_kare)

# Yöntem 3:
?sapply

sapply(x, FUN = "^", 2)

sapply(x, FUN = function(x){
  x * x
})
```

- `apply(...)` fonksiyonu

```
?apply

x <- matrix(c(1:25), nrow = 5, ncol = 5, byrow = TRUE)

# Sütun toplamları
colSums(x)
apply(x, MARGIN = 2, FUN = sum)

# Logaritmik sütun toplamları
colSums(log(x))
apply(x, 2, function(x){
  sum(log(x))
})
```

- `lapply(...)` fonksiyonu

```
?lapply

x <- matrix(c(1:25), nrow = 5, ncol = 5, byrow = TRUE)
x2 <- as.data.frame(x)

# Sütun toplamları
colSums(x)
lapply(x, FUN = sum)
lapply(x2, FUN = sum)
```

Örnek:

```
x <- matrix(c(1:25), nrow = 5, ncol = 5, byrow = TRUE)

myfun <- function(y, seviye = 8){
  sum(y >= seviye)
}

apply(x, 1, myfun)
apply(x, 2, myfun)
```

Fonksiyonlar

R programlama dilinde fonksiyonlar `function(<function arguments>){ <code lines> }` yapısı kullanılarak yazılır. Fonksiyonda kullanılacak olan parametreler `(<function arguments>)`, fonksiyonun kullanacağı kodlar ise `{ <code lines> }` içerisinde yazılır.

Basit bir fonksyion örneği:

```
myFun <- function(x, y){
  x / y
}

myFun(x = 10, y = 3)    # Bölme işlemi
```

```
## [1] 3.333333
```

Fonksiyon yazımında dikkat edilmesi gereken noktalar

Önemli: Fonksiyon yazımında aşağıdaki durumlardan kaçınmak gerekir.

- **for** döngüsü kullanımı (döngü sayısı çok fazla ise)
- Çok fazla sayıda içiçe **for** döngülerinin kulanımı
- Çok fazla sayıda içiçe **if** sınamalarının kullanımı
- **for** ile **while** döngülerinin içiçe kullanımı

Fonksiyonlarınız düzenli olması ve diğer programcılar tarafından rahatlıkla anlaşılabilmesi için evrensel kurallara uymak gerekir. R ile kod yazımında https://google.github.io/styleguide/Rguide.xml adresindeki kurallardan yararlanabilirsiniz.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand. (Martin Fowler et al, Refactoring: Improving the Design of Existing Code, 1999)

Soru 1: Sayısal değerlerden oluşan bir vektör için *ortalama, ortanca, standart sapma, en küçük* ve *en büyük* değerleri hesaplayan ve sonuçları bir liste içerisinde döndüren fonksiyonu yazınız.

```
set.seed(1)

# Ortalaması 20, standart sapması 3 olan normal dağılımdan 100 genişliğinde bir
# veri üretiliyor.
x <- rnorm(n = 100, mean = 20, sd = 3)

mean(x)
sd(x)
median(x)
min(x)
max(x)
```

```
descStats <- function(<function_arguments>){
  code lines ...
  code lines ...
}
```

Soru 2: [0, y] aralığındaki Fibonacci sayılarını hesaplayan fonksiyonu oluşturunuz. Fonksiyonda üst limit olan y sayısı tanımlanabileceği gibi, 0'dan itibaren hesaplanması istenilen Fibonacci sayılarının miktarı da girilebilir. Fonksiyonu her iki alternatif için çalışacak şekilde oluşturunuz.

Fonksiyon algoritması

- Fibonacci üst sınırını (y) veya toplam hesaplanması gereken Fibonacci sayısı miktarını (n) belirle.
- İlk iki Fibonacci sayısını 0 ve 1 olarak önceden tanımla.
- y ve n değerleri için mantıksal sınamaları belirle.
  - $y > 1$ olmalıdır.
  - $n > 1$ olmalıdır.
  - y ve n değerlerinden en az birisi girilmiş olmalıdır.
  - y ve n değerlerinin ikisi birlikte girilir ise öncelikli olarak y değeri dikkate alınacaktır.
- Döngüsel olarak Fibonacci sayılarını hesapla (`while` veya `for` kullanılacaktır)
- Fibonacci sayılarını vektör olarak kullanıcıya döndür.

```
fibonacci <- function(<function_arguments>){
  code lines ...
  code lines ...
}
```

