



# ile Veri Analizi

Dr. Bülent Çobanoğlu

cobanoglubulent[at]gmail.com

# İçindekiler

- rep(), rev(), seq(), all(), aby() fonksiyonları
- Döngüler
  - i. while , repeat, for
- Sonsuz Döngü
- next, break
- Apply ailesi
- istatiksel hesaplamalar (mean, median, summary )
- Rastgele Veri Üretimi
- Dağılımlar (rnorm, runif,...), z-score
- sample, subset()
- Fonksiyonlar
- Lambda fonksiyonlar
- Vektörler
- Eksik veri sorgulama,
- <<- operatörü ve scope kavramı
- İndis kavramı

# rep (tekrar fonksiyonu)

- Girilen bir ismi 5 kez alt alta yazdırma (döngüsüz)

```
1 ad = readline("Adınız.:")  
2 vec <- rep(ad, times = 5)  
3 cat(vec, sep="\n") |
```

```
Adınız.:ali  
ali  
ali  
ali  
ali  
ali
```

# rep (tekrar fonksiyonu)

```
1 # 1'i 5 kez tekrar et
2 vec1 <- rep(1, times = 5)
3 print(vec1)
4 # 1:3 vektörünü her öğeyi 2 kez tekrar et
5 vec2 <- rep(1:3, each = 2)
6 print(vec2)
7 # 1:4 vektörünü toplam uzunluğu 8 yapacak şekilde tekrar et
8 vec3 <- rep(1:4, length.out = 8)
9 print(vec3)
10 # 1'i 3 kez tekrar et, ancak son öğeyi tekrar etme
11 vec4 <- rep(1, times = 3, repeat.last = FALSE)
12 print(vec4)
13 # 1:4 vektörünü verilen kombinasyona (mehtere) göre tekrar et,
14 vec5 <- rep(1:4, c(2,1,2,1))
15 print(vec5)
```

```
[1] 1 1 1 1 1
[1] 1 1 2 2 3 3
[1] 1 2 3 4 1 2 3 4
- [1] 1 1 1
  [1] 1 1 2 3 3 4
```

# Ardışık Liste (sequence) Oluşturma

```
1 seq(1:5)
```

1 · 2 · 3 · 4 · 5

```
1 seq(from=1, to=5)
```

1 · 2 · 3 · 4 · 5

```
1 1:5
```

1 · 2 · 3 · 4 · 5

```
1 seq(from=1, to=5, by=2)
```

1 · 3 · 5

```
1 seq(1,5,2)
```

1 · 3 · 5

```
1 seq(1,5)
```

1 · 2 · 3 · 4 · 5

```
1 seq(from=5, to=1, by=-2)
```

5 · 3 · 1

```
1 seq(5,1,-2)
```

5 · 3 · 1

# `seq (from, to, by = step, length.out=len )`

- ▶ Ardışık sıralı sayılar üretir

```
seq(0, 10, length.out = 5)      # [1] 0 2.5 5 7.5 10
```

```
seq(stats::rnorm(10)) # [1] 1 2 3 4 5 6 7 8 9 10
```

```
seq(1, 9, by = 2)      # [1] 1 3 5 7 9
```

```
seq(9, 0, by = -3)     # [1] 9 6 3 0
```

```
seq(7) # same as 1:7, or seq_len(7)
```

```
# [1] 1 2 3 4 5 6 7
```

# seq (from, to, by = step, length.out=len )

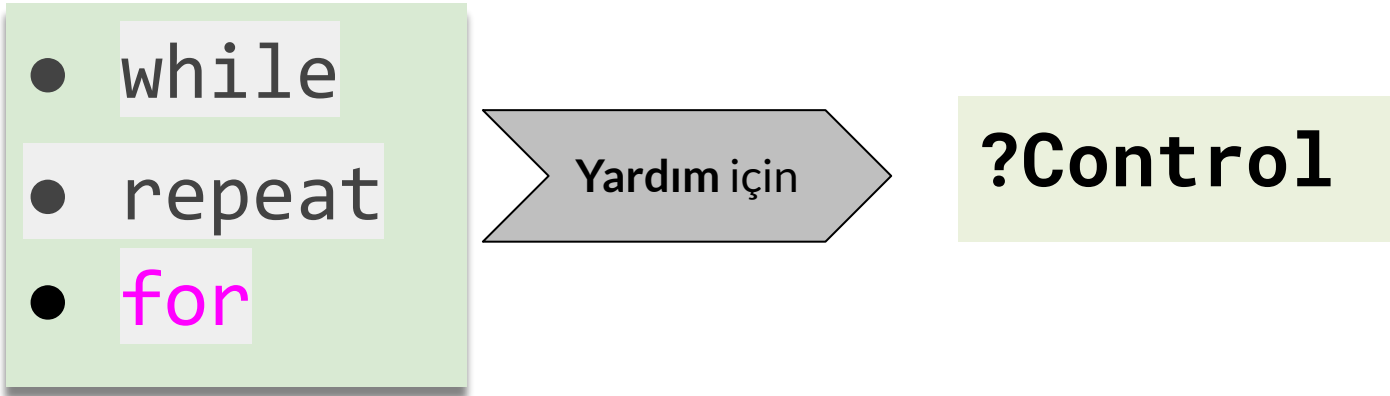
- ▶ Fonksiyon parametrelerinin sırası değişebilir mi?
- ▶ `z <- seq(to=40,by=5,from=5)` gibi.



Students choose an option

# Döngüler (Loops)

- ▶ Hemen hemen bütün programlama dillerinde olduğu gibi R dilinde de **farklı döngü yapıları** vardır. Bunlar;





# while

- İşlemlerin ne kadar **tekrarlanacağı**nın **koşulun doğru** ya da **yanlış** olmasına bağlı olduğu, **koşul geçerli olduğu sürece** işlemlerin tekrarlandığı döngülerdir.

```
while(koşul) {  
  #işlemler }
```

```
1 i <- 0  
2 while (i <= 4) {  
3 |   i <- i + 1  
4 |   cat(i, " ")  
5 }
```

1 2 3 4 5

# Task: while döngüsü

- A dan Z ye kadar İngilizce karakterleri ekranda gösteren programı while döngüsü ile kodlayalım. Programın örnek ekran çıktısı;

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

ASCII  
Tablosu

A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90



Students, write your response!

# Task : while

- ▶ A dan Z ye kadar İngilizce karakterleri ekranda gösteren programı while döngüsü ile kodlayalım.

```
1 i <- 65
2 while (i <= 90) {
3   cat(intToUtf8(i), " ")
4   i <- i + 1
5 }
```

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

# while (TRUE) ile Sonsuz Döngü

- Sonsuz döngü oluşturur. Döngü içerisindekiler koşulsuz çalıştırılır. Döngüden çıkış ancak **break** komutu ile döngü kırılarak gerçekleşir

**Kullanıcı Girişi Denetimi:** Doğru bir parola girene kadar kullanıcı girişini bekleten programı while True ile kodlayalım....

•• Parolanızı girin:

# while (TRUE)

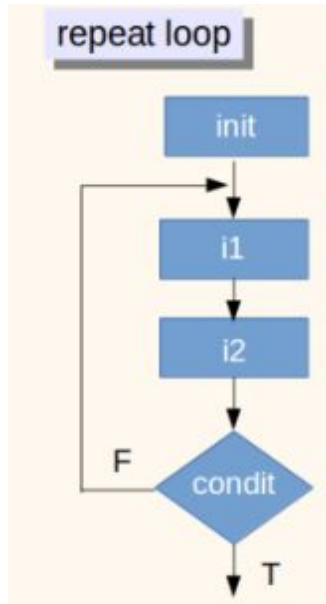
**Kullanıcı Girişi Denetimi:** Geçerli parola girilene kadar kullanıcı girişini bekleten programı while True ile kodlayalım....

```
my_password <- "sifre123"
while (TRUE) {
  girilen_password <- readline(prompt = "Parolanızı girin: ")
  if (girilen_password != my_password) {
    cat("Parola yanlış. Tekrar deneyin.\n")
    flush.console()
  } else {
    cat("Parola doğru. Giriş başarılı!\n")
    flush.console() # output bufferını temizler
    break          # Döngüden çık
  } #else sonu
} # while sonu
```

```
Parolanızı girin: ada
Parola yanlış. Tekrar deneyin.
Parolanızı girin: sifre123
Parola doğru. Giriş başarılı!
>
```

# repeat

- Sonsuz döngü oluşturur. Döngü içerisindekiler koşulsuz çalıştırılır. Döngüden çıkış ancak **break** komutu ile döngü kırılarak gerçekleşir



```
1 count <- 1
2 repeat {
3   cat("Say1: ", count, "\n")
4   count <- count + 1
5   if (count > 5) {
6     break # Döngüyü sonlandır
7   }
8 }
```

```
Say1: 1
Say1: 2
Say1: 3
Say1: 4
Say1: 5
```

# for

- Tekrar sayısının **baştan belli olduğu**, **ardışık eleman listesinin sırayla işleme konulduğu** döngü yapısıdır. for döngüsü **in** operatörünü kullanarak liste üzerinde ardışık (iteratif) ilerler.

```
for (eleman in liste)
{
    #işlemler
}
```



```
1 for(i in 1:5)
2 |   print(i)}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

# for - while eşdeğeri:

- 0' dan 20' e kadar ki (20 dahil) **çift sayıların toplamını (110)** ekranda gösteren programı hem **for** hemde **while** döngüsü ile kodlayalım.

```
T <- 0
for (a in seq(0, 20, by = 2)) {
  T <- T + a
}
# Döngü sonu
cat("Toplam = ", T, "\n")
```

Eşdeğeri

```
a = 0
T = 0 #Toplama değişkeni
while (a < 22) {
  T = T + a #Yığılmalı toplama
  a = a + 2
} #Döngü sonu
cat ("Toplam =", T)
```



# break deyimi

- Bir şarta bağlı olarak (**sonsuz döngü dahil**) herhangi bir döngüyü **sonlandırmak** için kullanılır.

```
1 for (i in 1:10) {  
2   if (i == 10) {  
3     break # 10 da çık  
4   }  
5   cat(i, " ")  
6 }
```

1 2 3 4 5 6 7 8 9

---

# next deyimi

- Bir şarta bağlı olarak alt satırları atlatarak **döngü başı yapmak** için kullanılır.
- **break** program akışını döngü sonrası komuta,
- **next** ise döngü başına getirir.

```
1 for (i in 1:10) {  
2   if (i %% 2 == 0) {  
3     next # Çift sayıları atla  
4   }  
5   cat(i, " ")  
6 }
```

---

1 3 5 7 9

# İç içe Döngüler (Nested Loops)

- ▶ Bir döngü yapısı içerisinde başka bir döngü kullanılarak **İç içe döngüler** oluşturulabilir. İç içe döngülere;

- ▶ **Zaman sistemi;**
- ▶ **Tablo şeklindeki (satır ve sütunlardan oluşan) çarpım tablosu gibi yapılar/desenler;**
- ▶ **Çok boyutlu diziler;**

örnek gösterilebilir.

# İç içe döngüler

```
1 calisan <- c("Ali", "Zeki", "Ayşe")
2 mod <- c("MUTLU", "MUTSUZ")
3 for (i in calisan){ # dış döngü
4 |   for (ii in mod){ # iç döngü
5 |       cat(i, ii, "\n")
6 |   }
7 }
```

```
Ali MUTLU
Ali MUTSUZ
Zeki MUTLU
Zeki MUTSUZ
Ayşe MUTLU
Ayşe MUTSUZ
```

# İç içe Döngüler (Nested Loops)

► **Task:**  Aşağıdaki çarpım tablosunu nasıl kodlarız?

1 * 1 = 1	2 * 1 = 2	3 * 1 = 3	4 * 1 = 4	5 * 1 = 5	6 * 1 = 6	7 * 1 = 7	8 * 1 = 8	9 * 1 = 9	10 * 1 = 10
1 * 2 = 2	2 * 2 = 4	3 * 2 = 6	4 * 2 = 8	5 * 2 = 10	6 * 2 = 12	7 * 2 = 14	8 * 2 = 16	9 * 2 = 18	10 * 2 = 20
1 * 3 = 3	2 * 3 = 6	3 * 3 = 9	4 * 3 = 12	5 * 3 = 15	6 * 3 = 18	7 * 3 = 21	8 * 3 = 24	9 * 3 = 27	10 * 3 = 30
1 * 4 = 4	2 * 4 = 8	3 * 4 = 12	4 * 4 = 16	5 * 4 = 20	6 * 4 = 24	7 * 4 = 28	8 * 4 = 32	9 * 4 = 36	10 * 4 = 40
1 * 5 = 5	2 * 5 = 10	3 * 5 = 15	4 * 5 = 20	5 * 5 = 25	6 * 5 = 30	7 * 5 = 35	8 * 5 = 40	9 * 5 = 45	10 * 5 = 50
1 * 6 = 6	2 * 6 = 12	3 * 6 = 18	4 * 6 = 24	5 * 6 = 30	6 * 6 = 36	7 * 6 = 42	8 * 6 = 48	9 * 6 = 54	10 * 6 = 60
1 * 7 = 7	2 * 7 = 14	3 * 7 = 21	4 * 7 = 28	5 * 7 = 35	6 * 7 = 42	7 * 7 = 49	8 * 7 = 56	9 * 7 = 63	10 * 7 = 70
1 * 8 = 8	2 * 8 = 16	3 * 8 = 24	4 * 8 = 32	5 * 8 = 40	6 * 8 = 48	7 * 8 = 56	8 * 8 = 64	9 * 8 = 72	10 * 8 = 80
1 * 9 = 9	2 * 9 = 18	3 * 9 = 27	4 * 9 = 36	5 * 9 = 45	6 * 9 = 54	7 * 9 = 63	8 * 9 = 72	9 * 9 = 81	10 * 9 = 90
1 * 10 = 10	2 * 10 = 20	3 * 10 = 30	4 * 10 = 40	5 * 10 = 50	6 * 10 = 60	7 * 10 = 70	8 * 10 = 80	9 * 10 = 90	10 * 10 = 100

# İç içe Döngüler (Nested Loops)

```
# Çarpım tablosu
for (i in 1:10) {
  for (j in 1:10) {
    cat(j, "*", i, "=", i * j, "\t")
  }
  cat("\n") # Satır sonuna geldiğinde 1 satır atla
}
```

1 * 1 = 1	2 * 1 = 2	3 * 1 = 3	4 * 1 = 4	5 * 1 = 5	6 * 1 = 6	7 * 1 = 7	8 * 1 = 8	9 * 1 = 9	10 * 1 = 10
1 * 2 = 2	2 * 2 = 4	3 * 2 = 6	4 * 2 = 8	5 * 2 = 10	6 * 2 = 12	7 * 2 = 14	8 * 2 = 16	9 * 2 = 18	10 * 2 = 20
1 * 3 = 3	2 * 3 = 6	3 * 3 = 9	4 * 3 = 12	5 * 3 = 15	6 * 3 = 18	7 * 3 = 21	8 * 3 = 24	9 * 3 = 27	10 * 3 = 30
1 * 4 = 4	2 * 4 = 8	3 * 4 = 12	4 * 4 = 16	5 * 4 = 20	6 * 4 = 24	7 * 4 = 28	8 * 4 = 32	9 * 4 = 36	10 * 4 = 40
1 * 5 = 5	2 * 5 = 10	3 * 5 = 15	4 * 5 = 20	5 * 5 = 25	6 * 5 = 30	7 * 5 = 35	8 * 5 = 40	9 * 5 = 45	10 * 5 = 50
1 * 6 = 6	2 * 6 = 12	3 * 6 = 18	4 * 6 = 24	5 * 6 = 30	6 * 6 = 36	7 * 6 = 42	8 * 6 = 48	9 * 6 = 54	10 * 6 = 60
1 * 7 = 7	2 * 7 = 14	3 * 7 = 21	4 * 7 = 28	5 * 7 = 35	6 * 7 = 42	7 * 7 = 49	8 * 7 = 56	9 * 7 = 63	10 * 7 = 70
1 * 8 = 8	2 * 8 = 16	3 * 8 = 24	4 * 8 = 32	5 * 8 = 40	6 * 8 = 48	7 * 8 = 56	8 * 8 = 64	9 * 8 = 72	10 * 8 = 80
1 * 9 = 9	2 * 9 = 18	3 * 9 = 27	4 * 9 = 36	5 * 9 = 45	6 * 9 = 54	7 * 9 = 63	8 * 9 = 72	9 * 9 = 81	10 * 9 = 90
1 * 10 = 10	2 * 10 = 20	3 * 10 = 30	4 * 10 = 40	5 * 10 = 50	6 * 10 = 60	7 * 10 = 70	8 * 10 = 80	9 * 10 = 90	10 * 10 = 100

# İstatiksel hesaplamalar

- Bir veri setinin merkezi eğilimini ölçmek için ortalama (mean), median (medyan) ve mod (mode) kullanılır.
- Bir veri setinin merkezi değişimini(yayılımını) ölçmek için standart sapma (standard deviation) ve varyans (variance) kullanılır.
- Bir veri seti hakkında özet istatistik bilgileri almak için **summary()** fonksiyonu kullanılabilir.

# İstatiksel hesaplamlar

```
data <- c(24, 16, 12, 10, 12, 28, 38, 12, 28, 24)
summary(data)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
10.0	12.0	20.0	20.4	27.0	38.0

```
mean(data)      # 20.4
```

```
median(data)    # 20
```

```
range(data)     #10 38
```



# Rastgele Sayı(Veri) Üretimi

- ▶ Rastgele veri üretimi için **runif()**, **rnorm()**, **sample()** gibi fonksiyonlar kullanılabilir.

Fonksiyon	Açıklama
<b>runif(1)</b> <b>runif(n, min=0, max=1)</b> <b>round(runif(6, min=1, max=49))</b>	<b>'0 - 1'</b> arasında <b>double</b> tipinde rastgele bir sayı üretir.  <b>'1 - 49'</b> arasında <b>int</b> tipinde rastgele 6 sayı üretir. (49 dahil).
<b>set.seed(sayi)</b>	# hep aynı sayılar üretilir  set.seed(5)  round(runif(6, min=1, max=49))
<b>rnorm(n, mean = 0, sd = 1)</b>	Belirtilen ortalama ve standart sapma değerlerine sahip n adet rastgele sayı üretir.  rnorm(100, mean = 50, sd = 10) #ortalama değeri 50 ve standart sapması 10 olan 100 adet rastgele sayı üretir;
<b>sample(liste, size=a)</b>	Listeden rastgele <b>a adet eleman</b> seçer, ve sıralar

# Sayısal Loto



- Sayısal lotonun **tekrarsız sayılardan oluşması istenmektedir**. Buna göre aşağıdaki kodda ne yapılırsa benzersiz kod üretilebilir?

```
sample(1:49, size=6, replace = TRUE)
```

8 · 6 · 8 · 13 · 29 · 10



Students, write your response!

# Sayısal Loto



- Sayısal lotonun tekrarsız sayılardan oluşması istenmektedir.

```
sample(1:49, size=6, replace = FALSE)
```

33 · 34 · 39 · 17 · 29 · 2

default halde ya  
da  
replace=FALSE  
yapılır.



Students, write your response!

# Zar atışı

- Bir zarın 10000 adet atışının rassal (uniform) dağılımı;

```
set.seed(1)
```

```
# 10,000 adet random değer üret (uniform dağılım)
```

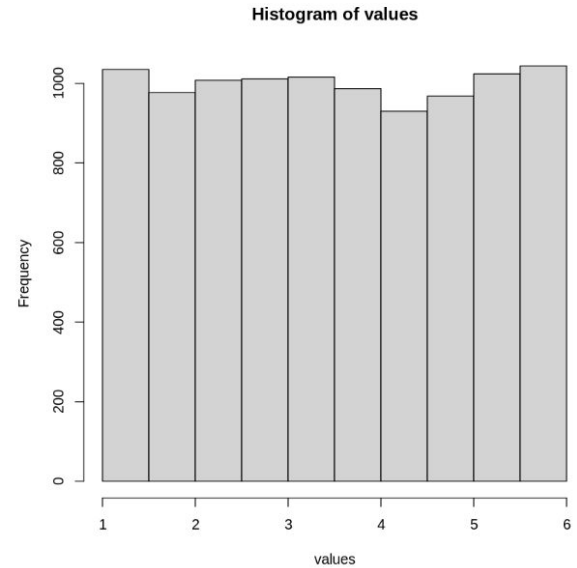
```
values <- runif(n=10000, min=1, max=6)
```

```
print("Zar atışı")
```

```
# histogramı çiz
```

```
hist(values)
```

[1] "Zar atışı"



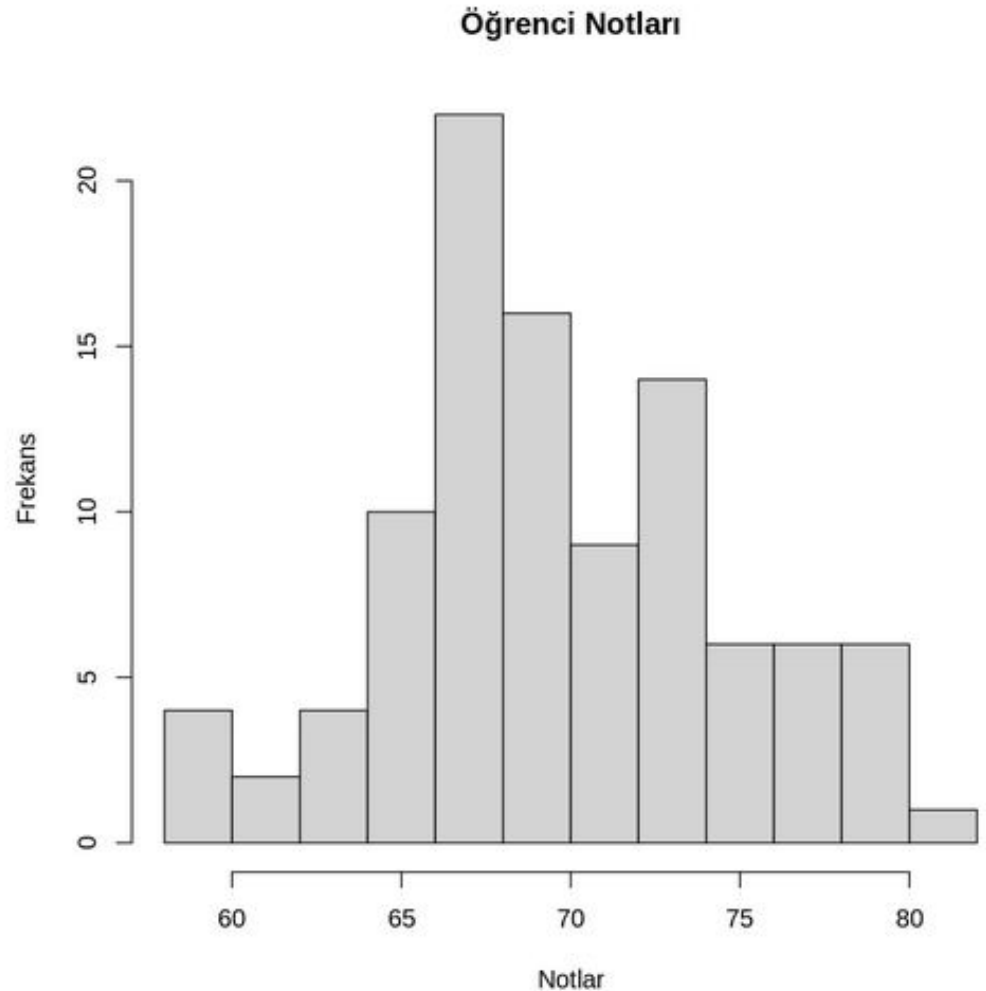
# Öğrenci notları

- Ortalama notu 70 ve standart sapması 5 olan rastgele 100 öğrenci notunun **normal dağılımını** histogram ile gösterelim.

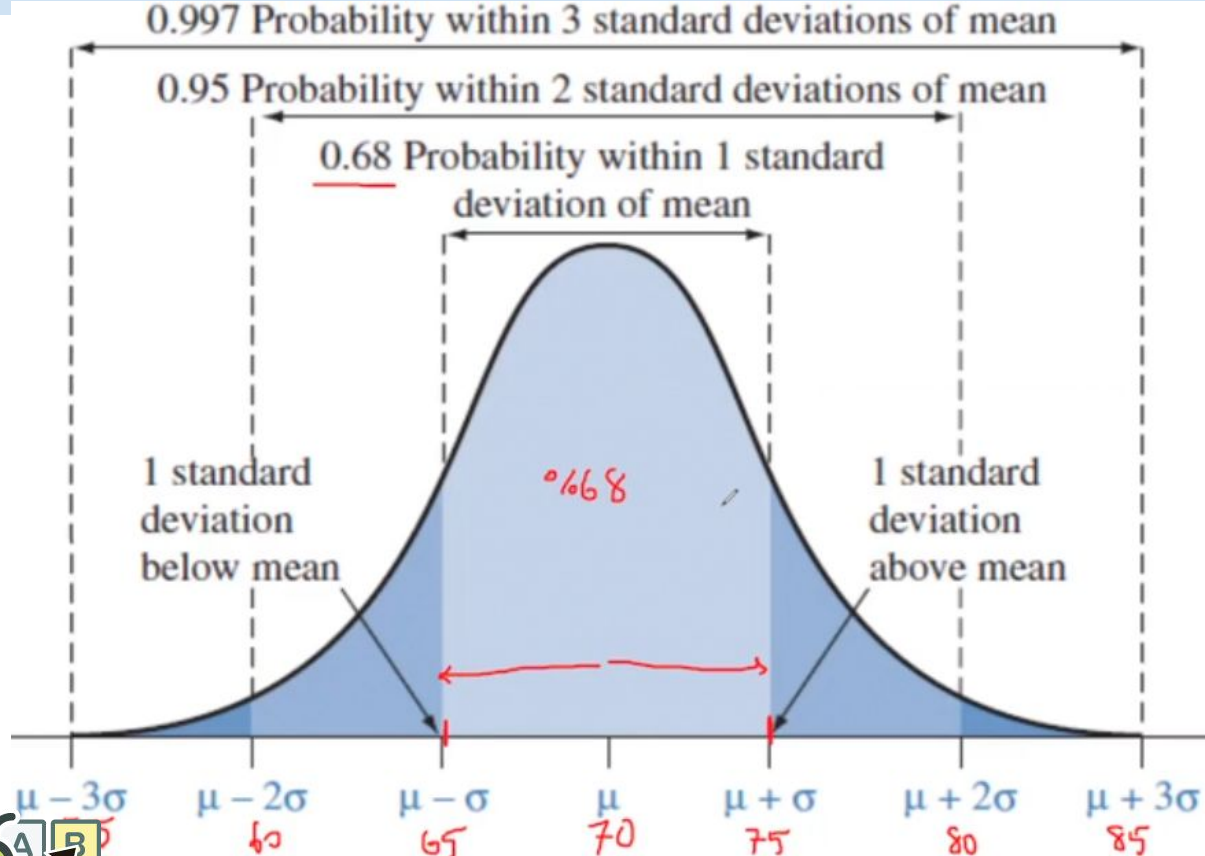
```
# # Normal Dağılım çan şeklinde özel bir yoğunluk eğrisidir.  
ortalama <- 70  
standart_sapma <- 5  
ogr_sayisi <- 100  
  
# Rastgele öğrenci notlarını oluştur  
notlar <- rnorm(ogr_sayisi, mean = ortalama, sd = standart_sapma)  
  
hist(notlar, main = "Öğrenci Notları", xlab = "Notlar", ylab = "Frekans")
```

# Öğrenci notları

- Ortalama notu 70 ve standart sapması 5 olan rastgele 100 öğrenci notunun **normal dağılımını histogram** ile gösterelim.



# Öğrenci notlarının normal dağılımı –



mean : 70

sd : 5

ise

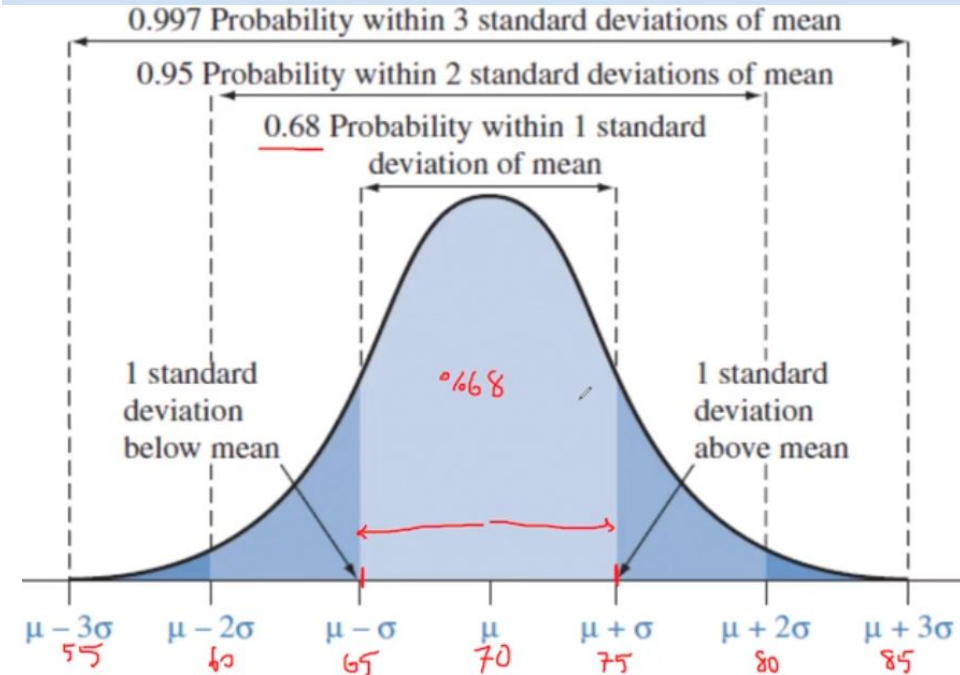
65-75 arası not  
alan yaklaşık kaç  
kişi vardır?

-Deneyisel kural!!--



Students choose an option

# Öğrenci notlarının normal dağılımı –



mean : 70

sd : 5

ise

65-75 arası not  
alan yaklaşık kaç  
kişi vardır?

```
length(notlar[notlar >= 65 & notlar <= 75]) # ~68
```



# z-score hesabı

- Z skoru, bir veri kümesindeki bilinen bir örneklemin ortalamadan kaç standart sapma yukarıda veya aşağıda olduğunu belirlemenizi sağlar.

$$Z = \frac{x - \text{mean}}{\text{standard deviation}}$$

- Ali sınavdan 78 almıştır. Ortalaması 70, standart sapması 5 olan bir not kümesine göre ortalamadan ne kadar uzaktan bir değer (z-score) almıştır?



Students, write your response!

# Veri Örnekleme: sample

► **sample(veri, size = 10)**

# Veri kümesinden 10 rastgele örnek al

► 20 adet 1-100 arasında üretilen notlardan;

○ 50 den yüksek olanların listesini;

○ 50 den yüksek olanlardan rastgele 5 örneği seçen

programı kodlayalım.

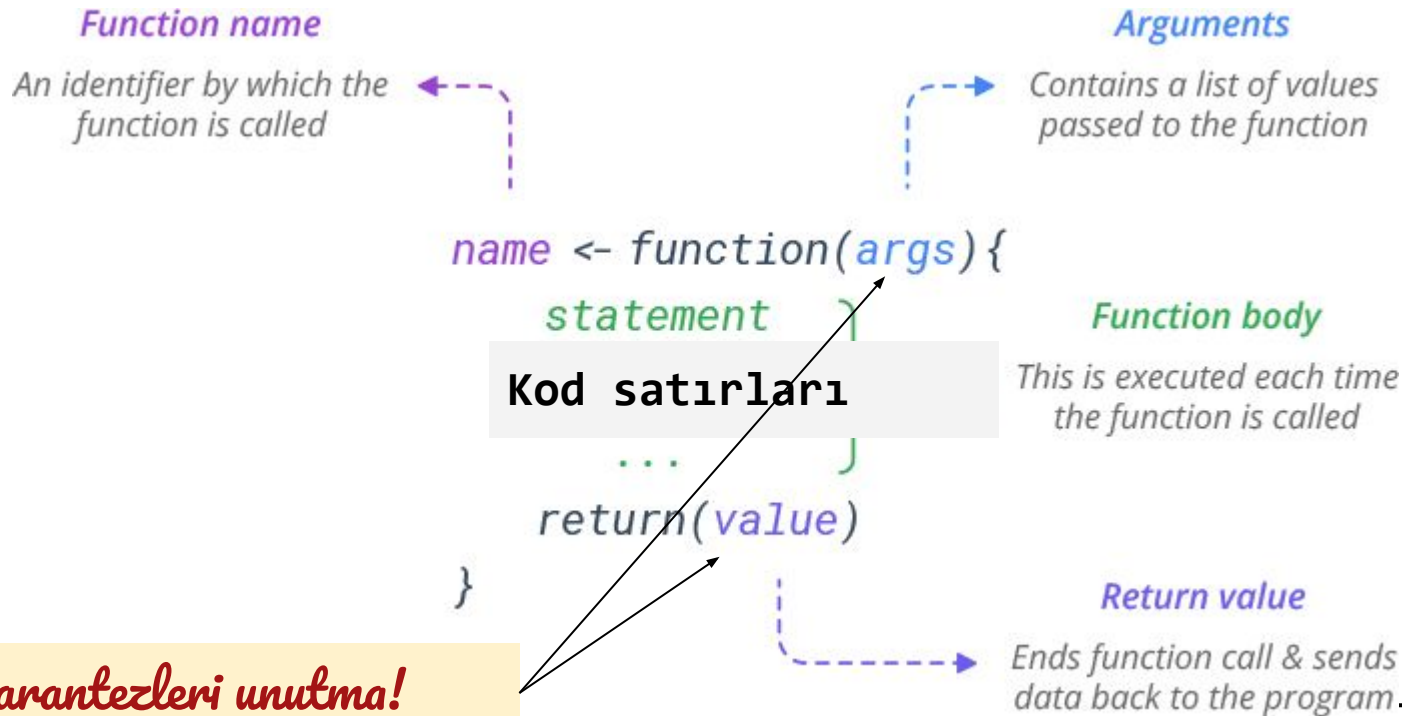
# Veri Örnekleme: resample

```
# sample örneği
notlar <- round(runif(20, min=1, max=100))
# 50 den yüksek olanlar
notlar_50 <- notlar[notlar>50]
print(notlar_50)
# rastgele 5 tanesi seç
notlar50 <- sample(notlar_50, size=5)
notlar50
```

```
[1] 57 72 70 99 99 96 54 58 69
70 · 99 · 99 · 69 · 54
```

# Kullanıcı tanımlı fonksiyonlar

- Kendi fonksiyonumuzu **'function'** deyimi ile tanımlayabiliriz;



# Kullanıcı tanımlı fonksiyonlar

- Geriye değer döndüren ve döndürmeyen fonksiyonlar;

```
# fonksiyon tanımlama
fx <- function(x){
|   return(x*x)
| }
}
```

```
#fonksiyon çağırma
fx(5)  # 25
```

```
selam <- function(isim) {
|   cat("Merhaba, ", isim, "!\n")
| }
}
```

```
selam("Dünya")
```

```
Merhaba,  Dünya !
```

---

# return kullanımı

- Fonksiyon tanımlanırken belirtilen girdilere '**parametre**', çağrılırken belirtilen girdilere ise '**argüman**' denir.

```
# fonksiyon tanımlama
fx <- function(x){
  ... f = x*x
  ... return(f)
}
```

```
#fonksiyon çağırma
fx(5) # 25
```

```
# fonksiyon tanımlama
fx <- function(x){
  |   return(x*x)
}
```

```
#fonksiyon çağırma
fx(5) # 25
```

```
# fonksiyon tanımlama
fx <- function(x){
  |   f = x*x
  |   return f
}
```


```
#fonksiyon çağırma
fx(5) # 25
```

```
Error in parse(text = x,
3:      f = x*x
4:      return f
               ^
```

*return (value) da  
parantezleri unutma!*

# Fonksiyon Tanımlama



- **Task:**  **hipotenus()** isimli bir fonksiyon tanımlayalım ve aşağıdaki değerler ile test edelim?

$$c = \sqrt{a^2 + b^2}$$

```
hipotenus(3,4)
```




Students, write your response!

# Fonksiyon Tanımlama

$$c = \sqrt{a^2 + b^2}$$




- **Task:**  **hipotenus()** isimli bir fonksiyon tanımlayalım ve aşağıdaki değerler ile test edelim?

```
hipotenus <- function(a,b){  
  c <- sqrt(a^2+b^2)  
  return(c)  
}  
#çağırılım  
hipotenus(3,4)
```



# Fonksiyon Tanımlama



- **Task:**  **carp()** isimli bir fonksiyon tanımlayalım ve aşağıdaki değerler ile test edelim?

```
carp(2,3)  
carp(-3,2.5)
```



Students, write your response!

# Fonksiyon Tanımlama

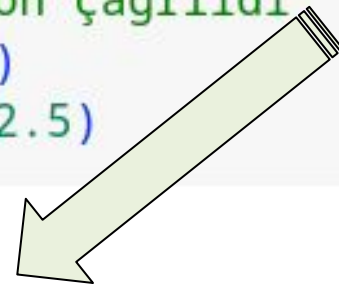


- **Task:** 🖐️ **carp()** isimli bir fonksiyon tanımlayalım


```
#fonksiyon tanımlandı  
carp <- function(a,b){  
  return(a*b)  
}
```

```
#fonksiyon çağrıldı  
carp(2,3)  
carp(-3,2.5)
```

6  
-7.5



# Fonksiyon argümanı vektör ise

- **Task:**  **isTekCift()** isimli bir fonksiyon tanımlayalım ve argüman olarak aldığı vektörün her bir değerine göre 'TEK' ya da 'ÇİFT' döndürelim.

```
# fonksiyonu çağıralım  
xx <- c(8, 3, 4, 5, 6, 7)  
yy <- isTekCift(xx)  
yy
```

'ÇİFT' . 'TEK' . 'ÇİFT' . 'TEK' . 'ÇİFT' . 'TEK'



Students, write your response!

# Fonksiyon argümanı vektör ise



```
isTekCift <- function(vec) {  
  result <- ifelse(vec %% 2 == 0, "ÇİFT", "TEK")  
  return(result)  
}
```

```
# fonksiyonu çağıralım  
xx <- c(8, 3, 4, 5, 6, 7)  
yy <- isTekCift(xx)  
yy
```

\_\_\_\_\_ 'ÇİFT' · 'TEK' · 'ÇİFT' · 'TEK' · 'ÇİFT' · 'TEK'

# Fonksiyonu özel operatöre dönüştürme



Aşağıdaki çıktıyı verecek

`'%strMul%'` isimli fonksiyonu nasıl tanımlarız?

```
"ali" %strMul% 5
```

```
# [1] "ali ali ali ali ali"
```



Students, write your response!

# Fonksiyonu özel operatöre dönüştürme



Aşağıdaki çıktıyı verecek

``%strMul%`` isimli fonksiyonu nasıl tanımlarız?

```
`%strMul%` <- function(dize, tekrarla) {  
  sonuc <- paste(rep(dize, times = tekrarla), collapse = " ")  
  return(sonuc)  
}
```

```
"ali" %strMul% 5 # [1] "ali ali ali ali ali"
```

# Anonim fonksiyonlar (Lambda)

▶▶ R da tek satırlık kısa **anonim (isimsiz)** fonksiyonlar tanımlanabilir.

▶▶ Kullanım şekli;

```
#fonksiyon tanımlandı  
carp <- function(a,b){  
  return(a*b)  
}
```

eşdeğeri

```
# İki sayıyı çarpan lambda fonksiyonu  
carp <- function(x, y) x * y
```

# Recursive fonksiyonlar



Kendi kendine atıf yapan fonksiyonlara rekürsif fonksiyonlar denir.

```
faktoriyel <- function(n) {  
  if (n == 0) {  
    return(1) # 0! = 1 : durdurma koşulu  
  } else {  
    return(n * faktoriyel(n - 1))  
  }  
}  
  
#fonksiyonu çağırılım  
faktoriyel(5)
```

120

eşdeğeri

factorial(5)



# İç içe Fonksiyon Tanımlama

- ▶▶ R da iç içe fonksiyon tanımlanabilir.
- ▶▶ Kullanım şekli;

```
# Dıştaki fonksiyon
dış_fonksiyon <- function(a, b) {
  # İçteki fonksiyon
  iç_fonksiyon <- function(x, y) {
    return(x * y)
  }
  sonuc <- iç_fonksiyon(a, b)
  return(sonuc)
}

# Dıştaki fonksiyonu kullanma
sonuc <- dış_fonksiyon(5, 3)
print(sonuc) # 15
```

# İç içe Fonksiyon Tanımlama



**Task:** 

Bu iç içe fonksiyon nasıl tek satıra indirgenebilir?

```
# Dıştaki fonksiyon
dış_fonksiyon <- function(a, b) {
  # İçteki fonksiyon
  iç_fonksiyon <- function(x, y) {
    return(x * y)
  }
  sonuc <- iç_fonksiyon(a, b)
  return(sonuc)
}

# Dıştaki fonksiyonu kullanma
sonuc <- dış_fonksiyon(5, 3)
print(sonuc) # 15
```



Students, write your response!

# İç içe Fonksiyon Tanımlama

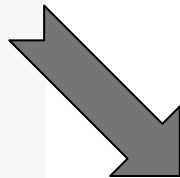


**Task:** 

Bu iç içe fonksiyon nasıl tek satıra indirgenebilir?

```
# Dıştaki fonksiyon
dış_fonksiyon <- function(a, b) {
  # İçteki fonksiyon
  iç_fonksiyon <- function(x, y) {
    return(x * y)
  }
  sonuc <- iç_fonksiyon(a, b)
  return(sonuc)
}
```

```
# Dıştaki fonksiyonu kullanma
sonuc <- dış_fonksiyon(5, 3)
print(sonuc) # 15
```



```
sonuc <- (function(a,b) (function(x, y) x*y) (a, b))(5,3)
print(sonuc) # 15
```

# Lokal değişkeni global yapmak



**Task:**  Bu iki program arasındaki fark nedir?

```
# R-> LEGB kuralı
var = 30
kapsayan_f<-function() {
  var = 20
  local_f<-function(){
    var = 10
    print(var)
  }
  local_f()
  print(var)
}
kapsayan_f()
print(var)
```

```
# R-> LEGB kuralı
var = 30
kapsayan_f<-function() {
  var <- 20
  local_f<-function(){
    var <- 10
    print(var)
  }
  local_f()
  print(var)
}
kapsayan_f()
print(var)
```

# Lokal değişkeni global yapmak



Bir yerel (local) değişkeni global kapsamda kullanmak için "<<-" atama operatörü kullanılır;

```
var = 30
kapsayan_f<-function() {
  var = 20
  local_f<-function(){
    var = 10
    print(var)
  }
  local_f()
  print(var)
}
kapsayan_f()
print(var)
```

```
[1] 10
[1] 20
[1] 30
```

```
var = 30
kapsayan_f<-function() {
  var <<- 20
  local_f<-function(){
    var <<- 10
    print(var)
  }
  local_f()
  print(var)
}
kapsayan_f()
print(var)
```

```
[1] 10
[1] 10
[1] 10
```

# Ders nasıl gidiyor? Duygunuz?



-1



0



1



Students, enter a number!

Pear Deck Interactive Slide  
Do not remove this bar

# Apply ailesi fonksiyonlar

- ▶ **for** döngüsü yerine **apply** ailesi fonksiyonları kullanılabilir;
- ▶ `apply(veri_yapisi, MARGIN, FONKSIYON, ...)`

Function	Input data type	Output data type
<b>apply</b>	<u>dataframe</u> or matrix or array (with margins)	vector, matrix, array, list
<b><u>lapply</u></b>	vector, list, variables in <u>dataframe</u> or matrix	list
<b><u>sapply</u></b>	vector, list, variables in <u>dataframe</u> or matrix	matrix, vector, list
<b><u>mapply</u></b> (multivariate <u>sapply</u> )	vector, list, variables in <u>dataframe</u> or matrix	matrix, vector, list

# for yerine apply kullanmak



**Task:** 📌 (1,2,3,4,5,6,7,8,9,10)



Bir vektörün her bir elemanını 1 artıran programını hem for döngüsü ile hem de **apply()** fonksiyonları ile yapınız?





# for yerine apply kullanmak



**Task:** 📌 (1,2,3,4,5,6,7,8,9,10)



Bir vektörün her bir elemanına 1 ekleyen program;

```
# eleman++ işlemi
vek <- (1:10)
new_vec <- vector()
for (eleman in vek) {
  new_vec <- append(new_vec, eleman+1)
}
print(new_vec)
```

[1] 2 3 4 5 6 7 8 9 10 11

---

# for yerine apply kullanmak



**Task:** 📌 (1,2,3,4,5,6,7,8,9,10)



Bir vektörün her bir elemanına 1 ekleyen program;

```
vek <- (1:10)
new_vec <- vector()
new_vec <- sapply(vek, function(x) x+1)
new_vec
```

2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · 11

# Veri Yapıları

- ▶ **Vector:** Aynı tip elemanlardan oluşan **tek boyutlu** bir dizi.
- ▶ **Matrix:** Aynı tip elemanlardan oluşan **iki boyutlu** bir dizi.
- ▶ **Lists:** Farklı tip elemanlardan oluşabilen çok boyutlu bir yapı.
- ▶ **Dataframe:** Farklı tip elemanlardan oluşabilen **iki boyutlu** bir yapı.

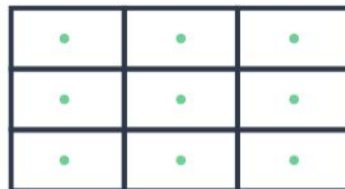
## Vector

1 Dimension | Same Data Type



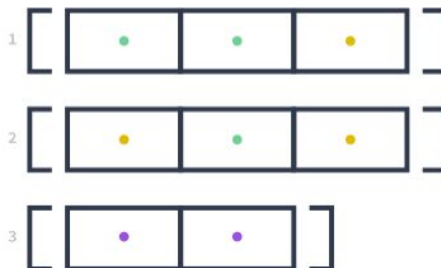
## Matrix

2 Dimensions | Same Data Type



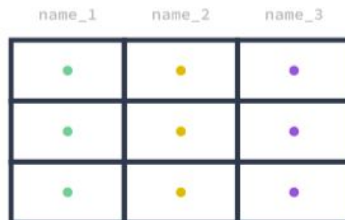
## List

Several Dimensions | Any Data Type



## Dataframe

2 Dimensions | Any Data Type



# Vektör

- ▶ **Vector:** Aynı tip elemanlardan oluşan **tek boyutlu** bir dizi.
- ▶ `c()` / `vector()` **# boş vektör**

```
str_ots <- c("1", "2", "3", "4")  
math_ots <- c(92L, 87L, 85L)  
exam_ots <- c(92, 90, 84, 95, 77, 92, 85)
```

```
print(str_ots)  
print(math_ots)  
print(exam_ots)
```

```
[1] "1" "2" "3" "4"
```

```
[1] 92 87 85
```

```
[1] 92 90 84 95 77 92 85
```

# Vektör

- **Task:** Aşağıdaki programların çıktısı nedir? *Düşünerek cevaplayalım.*  
*Kodlamayalım...*

```
str_notls <- c("1", "2", "3", "4")  
math_notls <- c(92L, 87L, 85L)
```

```
typeof(vector()) # tipi nedir?
```

```
bool_notls <- c(92, TRUE, FALSE, 95, 77L, 92, 85L) # tipi nedir?
```

```
combine <- c(str_notls, math_notls) # aynı tip mi?
```

```
str_int <- c("A", "B", 1, 3, 4, 5) # tipi nedir?
```

# Vektör

- Task: Aşağıdaki programların çıktısı nedir? *Düşünerek cevaplayalım.*

*Kodlamayalım...*

```
typeof(vector())
```

'logical'

```
str(vector("character", length=0))
```

chr(0)

```
bool_ots <- c(92, TRUE, FALSE, 95, 77L, 92, 85L)
print(bool_ots)
print(typeof(bool_ots))
```

```
[1] 92 1 0 95 77 92 85
[1] "double"
```

```
combine <- c(str_ots, math_ots)
print(combine)
print(class(combine))
```

```
[1] "1" "2" "3" "4" "92" "87" "85"
[1] "character"
```

```
str_int <- c("A", "B", 1, 3, 4, 5)
print(str_int)
print(typeof(str_int))
```

```
[1] "A" "B" "1" "3" "4" "5"
[1] "character"
```

# Bir vektörün uzunluğu

- Bir vektörün uzunluğu (eleman sayısı) **length()** fonksiyonu ile öğrenilir.

Boş string vektör; `vector("character", length=0)`

- Bir karakter dizisinin uzunluğu ise **nchar()** fonksiyonu ile öğrenilir.

```
vektor <- c("Ne", "olursan", "ol", "yine", "gel")
```

```
length(vektor) # vektörün eleman sayısı
```

5

```
nchar(vektor) # her bir karakterin uzunluğu
```

2 · 7 · 2 · 4 · 3

# Bir vektörü ters çevirme ve sıralama

- Bir vektörün sıralamak için **sort()**, ters çevirmek için ise **rev()** fonksiyonları kullanılır.

```
vec <- c(10, 20, 5, 30, 40, 5, 20, 160, 70)
reversed_vec <- rev(vec)
reversed_vec
```

70 · 160 · 20 · 5 · 40 · 30 · 5 · 20 · 10

```
] vec <- c(10, 20, 5, 30, 40, 5, 20, 160, 70)
sorted_vec <- sort(vec)
sorted_vec
```

5 · 5 · 10 · 20 · 20 · 30 · 40 · 70 · 160

```
► vec <- c(10, 20, 5, 30, 40, 5, 20, 160, 70)
sorted_vec <- sort(vec, decreasing = TRUE)
sorted_vec
```

160 · 70 · 40 · 30 · 20 · 20 · 10 · 5 · 5



# rank() ile elemanların sırası

- Bir vektördeki elemanların kendi içinde sırasını öğrenmek için **rank()** fonksiyonu kullanılır.

```
vec <- c(10, 20, 5, 30, 40, 5, 20, 160, 70)  
rank(vec)
```

```
3 · 4.5 · 1.5 · 6 · 7 · 1.5 · 4.5 · 9 · 8
```

```
beyin_dalgasi <- c("Alpha", "Delta", "Beta", "Theta", "Delta")  
rank(beyin_dalgasi)
```

```
1 · 3.5 · 2 · 5 · 3.5
```

# Tekrar eden elemanları temizleme

► **unique()** fonksiyonu ile tekrar eden veriler vektörden temizlenir.

```
#tekrarlı satırları temizle  
x<-c(5, 2, NA, 8, 12, 6, NA, 5, 12, 4, 3)  
unique(x)
```

5 · 2 · <NA> · 8 · 12 · 6 · 4 · 3

# Eksik veri sorgusu

- ▶ NA (Not Available) : Eksik / Kayıp veri
- ▶ NaN(Not-a-Number): Matematiksel işlemlerde anlamsız/tanımsız veri

```
# kayıp veri sorgusu:  
x<-c(5,2,NA,8,12,6)  
is.na(x)
```

FALSE · FALSE · TRUE · FALSE · FALSE · FALSE

```
x<-c(5, 2, NA, 8, 12, NaN, 6)  
is.nan(x)
```

FALSE · FALSE · FALSE · FALSE · FALSE · TRUE · FALSE

# NA değerlerin gösterimi

- ▶ NA (Not Available) : Eksik / Kayıp veri

```
x<-c(5,2,NA,8,12,6)
rank(x) # na.last=TRUE
```

$2 \cdot 1 \cdot 6 \cdot 4 \cdot 5 \cdot 3$

```
x<-c(5,2,NA,8,12,6)
rank(x, na.last=FALSE)
```

$3 \cdot 2 \cdot 1 \cdot 5 \cdot 6 \cdot 4$

```
x<-c(5,2,NA,8,12,6)
rank(x, na.last=NA) # NA'lar gözükmez
```

$2 \cdot 1 \cdot 4 \cdot 5 \cdot 3$

```
x<-c(5,2,NA,8,12,6)
rank(x, na.last="keep")
```

$2 \cdot 1 \cdot <NA> \cdot 4 \cdot 5 \cdot 3$

# all() ve any() fonksiyonları

- ▶ Bir vektördeki tüm elemanların verilen koşulu sağlayıp sağlamadığını test etmek için kullanılır;
- ▶ **all**(hepsi istenen koşulu sağlıyorsa TRUE, değilse FALSE)
- ▶ **any**(herhangi biri istenen koşulu sağlıyorsa TRUE değilse FALSE) sonucunu döndürür.

# all() ve any() fonksiyonları

- **all**(hepsi istenen koşulu sağlıyorsa TRUE, değilse FALSE)
- **any**(herhangi biri istenen koşulu sağlıyorsa TRUE değilse FALSE) sonucunu döndürür.

```
any(c(T, F, T, F, T, T, TRUE))
```

TRUE

```
all(c(T, TRUE, T, T, T, T, TRUE))
```

TRUE

```
x <- 1:10  
any(x > 8)
```


TRUE

```
all(x > 8)
```

FALSE

# all() ve any() fonksiyonları



- **Task:**  İki vektör elemanlarının birebir aynı olup olmadıklarını sorgulayan programı yazalım

```
# iki vektör aynı mı?
```

```
vek1 <- c(10, 11, 12, 13, 14)
```

```
vek2 <- c(10, 11, 12, 14, 13)
```



Students, write your response!

# all() ve any() fonksiyonları



## ► Task:

```
# Bu iki vektör aynı mı?
```

```
vek1 <- c(10, 11, 12, 13, 16)
```

```
vek2 <- c(10, 11, 12, 14, 13)
```

```
vek1==vek2 # birebir eşleştirme...
```

```
TRUE · TRUE · TRUE · FALSE · FALSE
```

```
all(vek1==vek2) # bütün elemanlar birbirine eşit mi?
```

```
FALSE
```

```
# Birinde olup diğerinde olmayanları seçmek için  
setdiff(vek1,vek2)
```



# İndis Kavramı

	1	2	3	← Indices
math_grades integer vector	92	87	85	← Values

- ▶ Vektörlerde, listelerde, dizi ve matrislerde değişken ismini takiben köşeli parantez `[indis_no]` içerisinde belirtilen numaraya **indis** adı verilir.
- ▶ Bir vektörün, karakter dizisinin veya listenin her bir elemanına indis numarası ile erişilir.
- ▶ *pozitif indis o elemanı; negatif indis o eleman haricindekileri verir.*

1	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	1

```
1 nums[5]
```

6

```
1 nums[-5]
```

10 · 9 · 8 · 7 · 5 · 4 · 3 · 2 · 1

# İndis Kavramı

*pozitif indis o elemanı; negatif indis o eleman haricindekileri verir.*

```
1 S = c("R", "S", "T", "U", "D", "I", "O")  
2 print(S[1])  
3 print(S[3])  
4 print(S[-3])
```

```
[1] "R"
```

```
[1] "T"
```

```
[1] "R" "S" "U" "D" "I" "O"
```

# İndis Kavramı

*Task: Çıktısı nedir?*

```
exam_ots <- c(92, 90, 84, 95, 77, 92, 85L)
```

```
exam_ots[c(1,3,7)]
```



Students, write your response!

# İndis Kavramı

*Vektör içerisinde logical sorgulama yapabiliriz.  
Bu programın çıktısı nedir?*

```
beyin_dalgasi <- c("Alpha", "Beta", "Theta", "Delta") # Vektör  
i <- c(FALSE, TRUE, FALSE, TRUE) # logical vector  
beyin_dalgasi[i] # TRUE'ları döndür  
beyin_dalgasi[!i] # FALSE'ları döndür  
beyin_dalgasi[i == FALSE] # FALSE'ları döndür
```



Students, write your response!

# İndis Kavramı

*Vektör içerisinde logical sorgulama yapabiliriz.*

*Bu programın çıktısı nedir?*

```
beyin_dalgasi <- c("Alpha", "Beta", "Theta", "Delta") # Vektör  
i <- c(FALSE, TRUE, FALSE, TRUE) # logical vector  
beyin_dalgasi[i] # TRUE'ları döndür  
beyin_dalgasi[!i] # FALSE'ları döndür  
beyin_dalgasi[i == FALSE] # FALSE'ları döndür
```

```
'Beta' 'Delta'  
'Alpha' 'Theta'  
'Alpha' 'Theta'
```

# subset() ile elemanı öğrenme

- Koşula uyan elemanların bir listesini almak için **subset()** metodu kullanılabilir;

```
x <- c(12, 3, 56, 4, 8, 9, 23, 44, 17)
subset(x, x>15 & x<30)
```

23 · 17

# subset() ile elemanı öğrenme

- İlk harfi 'B' ile başlayanlardan bir alt liste oluşturma;

```
beyin_dalgasi <- c("Alpha", "Beta", "Theta", "Delta", "Baba")  
subset(beyin_dalgasi, startsWith(beyin_dalgasi, "B"))
```

'Beta' · 'Baba'

# which() ile indisi öğrenme

- Bir elemanın indis numarasını öğrenmek için **which()** metodu kullanılabilir;

```
1 beyin_dalgasi <- c("Alpha", "Beta", "Theta", "Delta") # Vektör
2
3 indeks <- which(beyin_dalgasi=="Beta")
4 cat("Beta indeksi:", indeks, "\n")
5 cat("Alpha indeksi:", which(beyin_dalgasi=="Alpha"), "\n")
```

```
Beta indeksi: 2
Alpha indeksi: 1
```



---

---

# Teşekkürler!

## Bölüm Sonu

[bulendhoc@gmail.com](mailto:bulendhoc@gmail.com)

---