VIII. Magasabb rendű függvények

Bevezetés:

Az R programozási nyelvben vannak olyan függvények, amelyek legalább egy paramétere egy másik függvény. Ezeket a függvényeket magasabb rendű függvényeknek (higher-order functions) nevezzük. Alkalmazásuk jelentősen leegyszerűsítheti a programozást, áttekinthetővé, könnyen olvashatóvá téve a forráskódot. Az alábbiakban a magasabb rendű függvények közé tartozó `do.call` függvényt és az `apply` család függvényeit tekintjük át, a működésüket példákkal szemléltetve.

A `do.call` függvény:

A `do.call` függvénnyel egy másik függvényt hajthatunk végre annak argumentumán, amely lista típusú objektum.

Példák a függvény használatára:

1) Számítsuk ki a pi szinuszát először a `do.call` függvény használata nélkül, majd a `do.call` függvénnyel! Az R-ben a pi beépített konstansként érhető el, 3.141593 értékkel.

```
sin(pi)
[1] 1.224606e-16

do.call(sin, list(pi))
[1] 1.224606e-16
```

Ebben az esetben a `sin` a végrehajtandó függvény, a `list(pi)` pedig a `sin` függvény argumentumainak listája.

Ebben a példában még nem látszik a `do.call` függvény használatának előnye, ezért nézzük a következő példát!

2) Fűzzünk össze táblázatokat, méghozzá `data.frame` típusú változókat, amelyeket lista típusú objektumban tárolunk!

Példaképpen hozzuk létre az alábbi, három részlistából álló lista típusú objektumot és fűzzük ezek sorait össze a `do.call` függvénybe ágyazott `rbind` függvénnyel!

```
lista <- list(0)
lista[[1]] <- data.frame(
    "termes"=c("alma","korte","szilva"), "mennyiseg"=c(5,6,2))
lista[[2]] <- data.frame(
    "termes"=c("kukorica","buza","arpa"), "mennyiseg"=c(3,7,3))
lista[[3]]<- data.frame(
    "termes"=c("barack","dinnye","szolo"), "mennyiseg"=c(6,19,8))
lista
[[1]]
  termes mennyiseg
1   alma         5
2  korte         6
3 szilva         2

[[2]]
    termes mennyiseg
1 kukorica         3
2     buza         7
3     arpa         3
```

```
[[3]]
    termes mennyiseg
1   barack         6
2   dinnye        19
3    szolo         8
```

Alkalmazzuk a `do.call` függvényt!

```
tabla <- do.call(rbind, lista)
tabla
    termes mennyiseg
1     alma         5
2    korte         6
3   szilva         2
4 kukorica         3
5     buza         7
6     arpa         3
7   barack         6
8   dinnye        19
9    szolo         8
```

A `do.call(rbind,lista)` megoldás az `rbind(lista[[1]],lista[[2]],lista[[3]])` megoldással ekvivalens. Ha több tucat részlista állna rendelkezésre, áttekinthetetlenné válna az utóbbi módon megírt kód. A `do.call` függvény használata ráadásul gyorsabb is, tehát hatékonyabb.

Az R-ben a két függvény ekvivalenciáját az `all.equal` függvénnyel ellenőrizhetjük:

```
all.equal(do.call(rbind,lista), rbind(lista[[1]],lista[[2]],lista[[3]]))
```

Megjegyzés: A kód lefutási sebességét lemérhetjük a kód `system.time{(...)}` függvénybe ágyazásával. Másik lehetőség, ha a `microbenchmark` csomag `microbenchmark` függvényével végezzük el a sebességmérést. Utóbbi a `system.time` függvénnyel ellentétben többször fut le egymás után, így több futás eredményei alapján határozza meg a kód végrehajtásának átlagos sebességét.

Szintaktikája: `microbenchmark(kód1,kód2).`

Ellenőrizzük le a futási sebességet az előző példa esetén!

```
library(microbenchmark)

microbenchmark(do.call(rbind,lista),
rbind(lista[[1]],lista[[2]],lista[[3]]))

Unit: microseconds

min     lq         mean       median     uq         max         neval
185.802 189.0515   194.175    190.6015   192.501    287.201     100
186.501 189.9020   194.552    191.5515   193.401    323.901     100
```

Az `apply` függvénycsalád függvényei:

Az `apply` függvénycsalád függvényeivel valamilyen függvényt alkalmazunk összetett (többdimenziós) adattípuson.

Az `apply` függvény:

Az `apply` függvény sor- és oszlopadatokon dolgozik.

Szintaxisa:

```
apply(objektum, sor vagy oszlop szerinti művelet, függvény)
```

Példák a függvény használatára:

1) Tekintsük az R-ben alapértelmezetten elérhető, 32 sorból és 11 oszlopból álló `mtcars` táblázatot! Hajtsunk végre soronkénti összeadást, majd oszloponkénti átlagképzést! Előbbihez 1, utóbbihoz 2 írandó be az `apply` függvény argumentumába.

```
apply(mtcars, 1, sum)
apply(mtcars, 2, mean)
```

2) Az `apply` függvényben saját függvényt is alkalmazhatunk. Például írjunk függvényt, amely kiszámolja egy adatsor korrigálatlan empirikus szórását, majd a függvényt alkalmazzuk az `mtcars` táblázat oszlopaira.

```
szoras <- function(x) {
    sqrt(mean((x-mean(x))**2))
    }

apply(mtcars, 2, szoras)
```

Megjegyzés: Nem szükséges a függvényt elnevezünk, az `apply` függvény argumentumába közvetlenül is beilleszthetjük a saját függvényt. Ebben az esetben névtelen, úgynevezett *lambda* függvényt használunk.

A `tapply` függvény:

A `tapply` függvény egy tömb `(array)` adatait valamilyen faktor alapján csoportosítja, majd ezekre alkalmaz egy függvényt. Ez azt jelenti, hogy az `apply` függvénynél bemutatott összegképzéshez és átlagoláshoz képest bonyolultabb, valamilyen szempont szerint csoportosított adatokon is végrehajthatunk műveleteket.

Példa a függvény használatára:

Tekintsük ismét az `mtcars` táblázatot! Számítsuk ki az első oszlop (mpg) átlagait a kilencedik oszlop (am) szerint csoportosítva!

```
tapply(mtcars$mpg, list(mtcars$am), mean)
       0        1
17.14737 24.39231
```

Ugyanezt az eredményt az `aggregate` függvénnyel is megkaphatjuk:

```
aggregate(mtcars$mpg, list(mtcars$am), mean)

  Group.1        x
1       0 17.14737
2       1 24.39231
```

Az `aggregate` függvény azonban lassabban fut le a `tapply` függvénynél. Sebességüket mérjük meg a `microbenchmark` függvénnyel!

```
microbenchmark::microbenchmark(tapply(mtcars$mpg,  list(mtcars$am),  mean),
aggregate(mtcars$mpg, list(mtcars$am), mean))

Unit: microseconds

min        lq       mean      median    uq          max         neval
283.876    300.374  323.3457  321.707   333.0855    409.032     100
1311.857   1323.235 1407.0838 1330.062  1520.9240   2311.395    100
```

Az `aggregate` függvény `data.frame` típusú változót hoz létre, vagyis az eredményeket oszlopokban és sorokban helyezi el. A `tapply` függvény eredménye tömbben kerül tárolásra, amely gyorsabb, mint a `data.frame` típusú változóban való eltárolás, amely listákból épül fel.

A `tapply` függvény eredményeként kapott tömb elemei egyszerűbben hivatkozhatók, mint az `aggregate` függvényé. Például jelenítsük meg az 1 csoporthoz tartozó átlagot!

```
tapply(mtcars$mpg, list(mtcars$am), mean)[2]
       1
24.39231

aggregate(mtcars$mpg, list(mtcars$am), mean)[2,2]
[1] 24.39231
```

Megjegyzés:
Ha a kód alapján létrehozott objektum típusát, hosszát, dimenzióját stb. egymás után szeretnénk ellenőrizni, akkor a kód `class`, `length`, `dim` stb. függvényekbe ágyazása időigényes, hiszen a visszahívott kódban navigálnunk szükséges a függvénynevek lecserélése érdekében. A navigáció elkerülése céljából, az objektum megnevezése után használhatjuk a `magrittr` csomagban elérhető *pipe* operátort: `%>%`, amely matematikailag a függvénykompozíció műveletét jelenti. A pipe operátor tehát műveletek sorozatát hajtja végre egy objektumon.

Ha ez egy korábban megírt függvény, akkor ennek neve egyszerűen cserélhető, amennyiben másik műveletet szeretnénk végrehajtani.

A pipe operátor használatához töltsük be az R-ben a `magrittr` csomagot!

```
library(magrittr)
```

Példa típus- és hosszellenőrzés pipe operátor nélkül és pipe operátorral:

```
class(tapply(mtcars$mpg, list(mtcars$am), mean))
[1] "array"

length(tapply(mtcars$mpg, list(mtcars$am), mean))
[1] 2

tapply(mtcars$mpg, list(mtcars$am), mean) %>% class
[1] "array"

tapply(mtcars$mpg, list(mtcars$am), mean) %>% length
[1] 2
```

A műveletek sorozatának végrehajtása jól szemléltethető az alábbi példával. Írjuk az alábbi módon egymás után az utasításokat!

```
tapply(mtcars$mpg, list(mtcars$am), mean) %>% length %>% class
[1] "integer"
```

Először meghatározásra kerülnek az átlagok. Ezt követően az átlagokat tartalmazó tömb elemszámának, vagyis a kettes számnak a típusát (egész szám) határozzuk meg, nem pedig az átlagokat tartalmazó tömb típusát.

Ellenőrizzük a `tapply` és az `aggregate` függvénnyel megírt kód hasonlóságait / különbözőségeit az `all.equal` függvénnyel!

```
all.equal(tapply(mtcars$mpg, list(mtcars$am), mean), aggregate(mtcars$mpg,
list(mtcars$am), mean))

 [1] "Modes: numeric, list"
 [2] "Names: 2 string mismatches"
 [3] "Attributes: < Names: 2 string mismatches >"
 [4] "Attributes: < Component 1: Modes: numeric, character >"
 [5] "Attributes: < Component 1: target is numeric, current is character >"
 [6] "Attributes: < Component 2: Modes: list, numeric >"
 [7] "Attributes: < Component 2: Length mismatch: comparison on first 1 components >"
 [8] "Attributes: < Component 2: Component 1: Modes: character, numeric >"
 [9] "Attributes: < Component 2: Component 1: Lengths: 2, 1 >"
[10] "Attributes: < Component 2: Component 1: target is character, current is numeric >"
[11] "target is array, current is data.frame"
```

A `lapply` függvény: (list apply)

A `lapply` függvény végighalad egy lista vagy vektor minden elemén és függvényt alkalmaz azokra, amelynek eredményeként listát kapunk.

Példák a függvény használatára:

1) Töltsünk be az R-ben egyszerre több R csomagot a `lapply` függvénybe ágyazott `require` függvénnyel!

```
lapply(list("magrittr", "microbenchmark"), FUN=require, character.only=TRUE)
```

   Előbbivel kiválthatók az alábbi utasítások:

```
library(magrittr)
library(microbenchmark)
```

   Megjegyzés: a `character.only=TRUE` paraméter kulcsfontosságú, mert enélkül nem tudná változóként befogadni a `require` függvény a csomagneveket, a NonStandard Evaluation (NSE) miatt.

2) Állapítsuk meg egy beolvasandó állomány sorainak számát! Ehhez írjunk saját függvényt, amely a vizsgálandó állományt a `readLines` beépített R függvénnyel olvassa be!

```
getRowNumber <- function(fileInput) {
  length(readLines(fileInput))
}

lapply("elérési út/fájlnév", getRowNumber)
```

   A függvény eredményeképpen listában kapjuk vissza a sorszámokat.

A `sapply` függvény: (simplified apply)

A függvény vektor vagy lista típusú változón hajt végre egyszerűsítést és ugyanilyen típusú az eredmény is.

Példa a függvény használatára:

A `lapply` függvényhez írt második példát átírhatjuk `sapply` függvénnyel is. Ekkor az eredmény vektor típusú objektum, amely előnyös, hiszen a `lapply` függvény eredményeként lista típusú objektumot kapunk, így a numerikussá alakításához az eredményen végrehajtandó az `unlist` függvény.

```
sapply("elérési út/fájlnév", getRowNumber)
```

Felhasznált szoftver és kiegészítő csomagok:

R Core Team (2018): R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL: https://www.R-project.org/

Bache, SM, Wickham, H (2014): magrittr: A Forward-Pipe Operator for R. URL: https://CRAN.R-project.org/package=magrittr

Mersmann, O (2019). microbenchmark: Accurate Timing Functions. R package version 1.4-7. https://CRAN.R-project.org/package=microbenchmark