Bevezetés az R használatába

© Tóthmérész Béla

Tartalomjegyzék

1. Az R programkörnyezet használata

Virtues of a good programmer (A good *postmodern* programmer ...)

◇ Laziness • Impatience ◇ Hubris

Az igazi programozó tulajdonságai (Igazi *posztmodern* programozó ...)

◇ Lusta • Türelmetlen ◇ Hajlamos a világmegváltásra

– IMG 0218 2004 useR! Conference, Wienna, Austria

1.1. Az R alapjai

Az R programnyelv és programkörnyezet előnyei

Az R egy integrált programkörnyezet adatfeldolgozáshoz, számolási eljárások kidolgozásához és grafikus adatmegjelenítéshez. Az a kifejezés, hogy az R egy *programkörnyezet* azt jelenti, hogy nem csak specifikus számolási eljárásokat használhatunk, hanem egy olyan lehetőséget kapunk, ami tetszőleges célú további számolási algoritmusok kifejlesztésére, programozáshoz és a kidolgozott algoritmusok tesztelésre is alkalmas. Azaz az R egy programozási nyelv és programkörnyezet is egyúttal.

Számos egyéb tulajdonsága mellett az alábbi előnyös tulajdonságait említem meg felhasználói és adatfeldolgozási, illetve programozói és programfejlesztői szempontból. A sor minden bizonnyal tovább bővíthető egyre mélyebbre ásva az R világába.

Felhasználói és adatfeldolgozási szempontok

- Ingyenes, korlátozás nélkül használható (*GNU license*) a nem üzleti célú alkalmazások során.
- Könnyen telepíthető az R honlapjáról szabadon letölthető telepítőprogram révén.
- Az R használatát oktató és segítő, több ezer oldalnyi terjedelmű dokumentáció tölthető le a világ számos nyelvén az R honlapjáról és más internetes forrásokból.
- Igen aktív levelezőlista segíti a felhasználókat gondjaik megoldásában.
- Negyedévente megjelenő kiadvány (*Newsletter*) segíti a felhasználókat a R nyújtotta lehetőségek jobb megismerésében.

- A `CRAN` archívumban több mintegy 400 adatfeldolgozási és egyéb számolási eljárás áll a felhasználók rendelkezésére széleskörű dokumentációval együtt, *kiegészítő programcsomagok* (*library*) formájában. Ezen túlmenően a fejlesztők saját honlapjairól további R bővítések tölthetők le, amelyek még nem kaptak helyet a `CRAN` archívumban.
- Számos, a szakterületén vezető folyóirat esetében az újonnan kidolgozott módszerek alkalmazásához szükséges eljárásokat R segítségével teszik hozzáférhetővé a kutatók számára. Például az *Ecology* vagy a *Journal of American Statistical Association* (JASA) honlapján számos ilyen új eljárás látott napvilágot. A molekuláris biológiában (elsősorban a genomikai kutatásokhoz kapcsolódóan) külön internetes honlapot tartanak fenn, kapcsolódva az R-hez (*Bioconductor*), hogy segítsék a statisztikai eljárások széleskörű elterjedését.

Programozói és programfejlesztői szempontok

- Az R-nek kiváló beépített segítségnyújtó és a felhasználást segítő rendszere van, aminek segítségével gyorsan kaphatunk részletes információkat az egyes eljárások részleteiről a munka közben.
- Az R interaktívan használható, ami az alkalmazások és a programfejlesztések során is előnyös, mert gyorsítja a munkát.
- Az R vektorizált.
- Az R objektumorientált.
- Igen gazdagok és sokrétűek az adatstruktúrák.
- Hatékony adatkezelési és adattárolási lehetőségek állnak rendelkezésre.
- Az R nyílt forráskódú renszer.
- Az R-hez készített kiegészítő programcsomagok forráskódjai is nyilvánosak.
- Lehetőség van a `C/C++` és `Fortran` programkódok beillesztésére az R programokba.
- Az R hatékonyan támogatja a grafikus alkalmazásokat.

Az R program és programkörnyezet installálása

Az R programnyelv és programkörnyezet szabadon letölthető az internetről az R Project (egészen pontosan: R *Project for Statistical Computing*) honlapjáról. Az install egyetlen `exe` fájl formájában tölthető le (mintegy 20 Mb méretű) és a letöltés után elindítva minden szükséges installálási feladat egyszerűen elvégezhető. A gyakran használt platformok mindegyikére hozzáférhető a program.

Az installálás menetét tömören ismertetem. Részletesebb útmutatások a függelékben találhatók. A szokásos web böngészőnket használva nyissuk meg az R Project honlapját (`http://www.r-project.org/`). Kattintsunk a `CRAN` linkre a képernyő bal oldalán lévő menüben. Ez egy listát ad azokról a szerverekről, ahonnan az R letölthető. Célszerű azt választani, amely a legközelebb van; azaz a magyarországi vagy valamely közeli, ausztriai egyetem szerverét célszerű választani. Ezután ki kell választani a számítógépünkön futó (vagy ha rosszul választottunk, akkor időnként lefagyó) operációs rendszert. Az installálást vázlatosan ismertetjük a Windows esetében, feltételezve, hogy más operációs rendszerrel rendelkező olvasónak van annyi számítástechnikai kultúrája, hogy önállóan is tud egy programot installálni.

Az egérrel kattintsunk a "*Windows (95 and later)*" link-re, majd a "*base*" feliratra, mivel ott található az alapvető eljárásokat tartalmazó programcsomag. Töltsük le az "*R-2.2.1-win32.exe*" fájlt, vagy értelemszerűen az ennek megfelelő fájlt, ha már van újabb verzió. A letöltés befejezése után keressük meg a letöltött "*R-2.2.1-win32.exe*" fájlt a számítógép merevlemezén és kattintsunk rá. Ekkor elindul az R programnyelv és programkörnyezet installálása. Ha bármilyen technikai gond adódik, akkor kérjük segítséget.

Indítsuk el a programot

A sikeres installálás után megjelenik a számítógépünk desktop-ján az R ikonja, ami egy nagy R betű, alatta a program verziószámával. A könyv írásakor "R 2.0.1" az aktuális verziószám. Két dolog rögtön látható. Az egyik, hogy minden bizonnyal igényes programról van szó, mert a verziószám nem mutatja a kereskedelmi forgalomban kimagasló áron kapható, nagy költséggel reklámozott, de lényegében minőségbiztosítás nélküli programokat jellemző "számháború" (értsd verziószám háború) jegyeit. Azaz nem 21. század a verziószám, pedig milyen jól csengene ... A másik dolog, ami azonnal látszik a verziószám alapján, hogy a számozás a Unix-os rendszereknél szokásos konvenciót követi. Ha a verziószám utolsó jegye páros, például 2.0.0, akkor beta-változatról van szó, azaz egy olyan stabil, megbízható, új verziójáról a programnak, ami még nem végleges, mert még széleskörű, nyilvános tesztelésnek vetik alá. Amikor a verziószám utolsó számjegye páratlan, akkor az a program végleges változatát jelöli (például 2.0.1).

Az R ikonjára kattintva elindul az R programozási környezet. Ez, eltérően az egér-orientált kattingatós, zenélős, multimédiás rémségektől egy programozási környezet, ami azt jelenti, hogy a képernyő tetején csak viszonylag kevés legördülő menüt látunk, viszonylag kevés opcióval. Ezek a programozási környezet használatát támogatják és nem közvetlenül a statisztikai vagy egyéb célú számolások elvégzését. A statisztikai vagy egyéb célú számolásokhoz bizonyos szintű, alapfokon rendkívül egyszerű programozási ismeretre van szükségünk. Némileg egyszerűbben fogalmazva: be kell gépelnünk a parancs nevét. Az ENTER billentyű megnyomása után a parancs azonnal végre is hajtódik és megjelenik az

eredmény. Ebből rögtön láthatjuk, hogy a programozási környezet interpreteres programozás nyelvet takar, azaz nincs szükségünk fordítóprogramra, amivel le kellene az általunk írt programot fordítani valamilyen a számítógép által is megérthető formára, mielőtt azt futtatni szeretnénk. Ez rendkívül fontos azok számára, akik adatfeldolgozási célból kívánják az R-t használni és viszonylag kevés programozói vagy számítógépes ismerettel rendelkeznek.

Az R programozási környezet elindulásakor a 1. ábrán olvasható üzenet jelenik meg a képernyőn, ami a szerzői jogra vonatkozó információt tartalmazza, illetve azt, hogy hogyan kérdezhetjük meg a rendszertől a programra való irodalmi hivatkozás módját, hogyan indíthatjuk el a *demo*-t, hogyan kérhetünk segítséget, végül hogyan léphetünk ki a programból. Ha valaki nem kívánja begépelni a `q()` parancsot, ami a `quit` (kilépés) rövidítése, akkor kiléphet a legördülő menü "*File*" és "*Exit*" gombjaira történő kattintások segítségével, az egér intenzív és önelégült rángatása közepette. Bármelyik módját is választjuk a kilépésnek, a rendszer felteszi az alábbi kérdést: "*Save workspace image?*" Javaslom, hogy a "*No*" opciót válasszuk. Ellenkező esetben a rendszer menteni az összes adatot és segédprogramot is, amivel dolgoztunk és a következő indításkor készségesen visszatölti a memóriába. Ez igen hasznos a gyakorlott felhasználók számára, mert a folyamatban lévő adatfeldolgozást nem kell elölről kezdeni, ha valamilyen okból félbe kell szakítani a munkát, hanem ott folytathatjuk, ahol félbehagytuk. Ha azonban nem ez a helyzet, hanem egyszerűen csak lustaságból, figyelmetlenségből vagy tudatlanságból mindig a "*default option*"-t választjuk, ami "*Yes*", akkor előbb-utóbb annyi "történelmi emlékkel" lesz tele a számítógépünk memórája, hogy használatra gyakorlatilag alkalmatlanná válik.

```
R : Copyright 2005, The R Foundation for Statistical Computing
Version 2.2.1  (2005-12-20 r36812) ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY. You are
welcome to redistribute it under certain conditions. Type
'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors. Type
'contributors()' for more information and 'citation()' on how to
cite R in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help. Type 'q()' to
quit R.
```

1. ábra. Az R bejelentkező képernyőjének szövege.

1.2. Az első szárnypróbálgatás

Próbáljuk meg szóra bírni a rendszert, miután sikeresen elindítottuk. Az R környezetet használva a prompt jele ">". Ez jelzi, hogy beírhatjuk a parancsot, amit szeretnénk, hogy végrehajtson a program. A parancs begépelése után meg kell nyomni az ENTER billenytűt, hogy a prarancsot végrehajtsa a program. Ha a beírt parancs olyan komplex, hogy nem fér ki egyetlen sorban, akkor ezt a rendszer a következő sor elején megjelenő "+" jellel jelzi. Ez ebben az esetben nem műveleti jel.

Próbálkozzunk az R statisztikai környezet használatával. Írjuk be a ">" promthoz, hogy

```
> 5+2
[1] 7
```

Első látásra az eredmény kicsit meglepő módon a "[1]" jel után jelenik meg, ami az R esetében mindig az eredményt jelzi. Pontosabban, az eredményt jelző prompt mindig a "[ ]" jel az R-ben. A benne lévő szám a programnyelv vektorizált jellegével függ össze. Ha egyetlen szám az eredmény, akkor a promptként szolgáló szögletes zárójelben lévő szám "1". Olyan esetben, amikor az eredmény nem fér el egyetlen sorban, akkor a "[ ]" jelben az a szám jelenik meg, ahányadik a második sorban elsőként álló szám. Ezt egy példával illusztráljuk. Gépeljük be, hogy 1:30, ami azt jelenti az R számára, amint azt a későbbiekben látni fogjuk, hogy írja ki az egész számokat 1-től 30-ig!

```
> 1:30
 [1]  1  2 3  4 5 6 7  8  9 10 11 12 13 14 15 16 17 18 19
[20] 20 21 22 23 24 25 26 27 28 29 30
```

A képernyőn nem férnek el az egész számok 1-től 30-ig el egyetlen sorban és a "[20]" prompt jelzi, hogy a második sor első száma a huszadik szám.

A négy alapművelet jele az R-ben az általánosan elterjedt szokásnak megfelelő: összeadás: +, kivonás: -, szorzás *, osztás: /. A parancsok esetén a műveleti jelek előtt vagy után beírt szóköz nem befolyásolja az eredményt. Próbáljuk is ki ezeket:

```
> 6+17
[1] 23

> 7-19
[1] -12

> 6*7
[1] 42

> 4/3
[1] 1.333333
```

A hatványozást kétféle módon is jelölhetjük: ** vagy ^ jellel. Például:

```
> 2**3
[1] 8

> 2^3
[1] 8
```

A műveletek precedenciája (azaz végrehajtásuk sorrendje) is a szokásos módon értelmezett, amelyet zárójelekkel befolyásolhatunk.

```
> 4+5*2
[1] 14

> (4+5)*2
[1] 18

>  6/3^2
[1] 0.6666667

> (6/3)^2
[1] 4
```

Értékadás

Az R-ben az értékadás rendkívül logikus módon a "<-" és "->" nyilakkal történik, ahol a nyilak iránya jelzi az értékadást. Azaz az "x <- 9" azt jelzi, hogy az x változó értéke legyen 9. Korábbi verziókban az S/S++ programcsomaggal való kompatibilitás miatt az "_" hozzárendelési oprátor is használható volt. Ezt akkor fontos, ha valaki régebbi programkódot vagy eredetileg S-hez kifejlesztett programkódot kíván R-ben futtatni, mert emiatt hibaüzenetet kap. A pobléma könnyen megoldható a "_" jelek "<-"-re cserélésével. Ugyanakkor a dolog egy kicsit bonyolultabb, mert x_3 egy változónak a neve az 1.9 vagy annál nagyobb verziószámú R-ben, ugyanakkor a korábbiakban vagy S++-ban ez egy "x<-2" értékadásnak felel meg. Az x változónak adhatunk értéket az "=" egyenlőség jelet használva. Ez szintén egyfajta kompromisszum eredménye. A következetesség szószólói a "<-" és "->" jelekkel történő értékadás mellett kardoskodnak, mert a használata keveseb félreértésre adhat okot. Használható a lényegesen hosszabb assign() hozzárendelési operátor is. Tehát az alábbi

```
assign("x", 7)
x <- 7
x = 7
7 -> x
```

értékadások egyenértékűek.

Az `x` változó nem csak numerikus értéket, hanem karakteres értéket (stringet) is felvehet. Ekkor a karakter soroztatot idézőjelek közé kell írni:

```
> x <- "Hahó"
> "Hahó" -> x
> x
[1] "Hahó"
```

Az `x` változó értéke egy objektumként tárolódik a memóriában. Az `objects()` parancs segítségével lekérdezhetjük, hogy milyen objektumok vannak betöltve az R memóriájában. Ha a memóriában túl sok vagy túl nagy helyet lefoglaló objektumok vannak, akkor ezeket célszerű törölni, ha már nam használjuk. Ezt a `remove(x)` parancs segítségével tehetjük meg. Ezt a parancsot rövidíthetjük is `rm(x)` formában.

Logikai operátorok

A logikai változók két értéket vehetnek fel: `TRUE` (`igaz`) és `FALSE` (`hamis`). A jelölések a programozási nyelvekben szokásos konvenciókat követi: nagyobb `">"`, kisebb `"<"`, nagyobb egyenlő `">="`, kisebb egyenlő `"<="` egyenlő `"=="`, nem egyenlő `"!="`.

Az alábbi példák szemléltetik a logikai operátorok használatát:

```
> A <- 5
> B <- 9
> A < B
[1] TRUE
> A < A
[1] FALSE
> A <= B
[1] TRUE
> B == B
[1] TRUE
> A != B
[1] TRUE
```

Be- és kimenet, és a `print()` parancs haszálata

A R programozói környezettel történő ismerkedés elején célszerű olyan módon dolgozni, hogy a tananyag példáit, amennyiben elektronikus formában rendelkezésünkre állnak, átmásoljuk az asztal (*desktop*) memóriapufferén keresztül közvetlenül az R munka-ablakába vagy egy saját szövegfájlba. Hasonlóképpen a futtatások eredményei is kimásolhatók az R munka-ablakából a szokásos módon. Azaz úgy, hogy egérrel kiszínezzük az átmásolni kívánt részeket. Majd a másolás és a beillesztés parancsait használva az operációs rendszernek a kívánt

szövegszerkesztőben dokumentálhatjuk próbálkozásaink eredményét. Amennyiben valamilyen grafikus eredményt kapunk, akkor jobb gombbal az R által rajzolt ábrába kattintva felajánlja a rendszer, hogy átmásolhatjuk vagy diszkre írhatjuk az ábrát. Tehát ez is könnyen és kényelmesen dokumentálható. Ha már van némi gyakorlatunk az R használatában, akkor célszerű az alábbi bekezdésekben leírtak szerint eljárni.

Ha az adatfeldolgozás menetét vagy valamilyen számolási algoritmust gyakran használunk, akkor célszerű az egy formattálás nélküli szövegfájlban dokumentálni. (Az, hogy mi nem látunk egy szövegben formázást a képernyőn, az nem jelenti azt bizonyos programok esetén, hogy ilyen nincs is benne!) Ha a szövegfájl neve `"huha.txt"`, akkor a `source("huba.txt")` parancs segítségével tölthetjük be, ha a fájl a default alkönyvtárban van. A `source` parancs szintén megtalálható a "*File*" legördülő menüben.

Gyakran hasznos a `sink` parancs, amivel átirányíthatjuk a kimenetet a képernyőről egy fájlba; például `sink("proba.txt")` formában használhatjuk. A kimenet visszairányítása a képernyőre a `sink()` parancs kiadásával történik.

A `print()` parancs haszálata. Az R a változó nevének beírásával kiírja a képernyőre. Hasonlóképpen egy parancs kiadásakor az eredmény szintén a monitoron jelenik meg, ha azt nem irányítjuk át értékadással valamely változó értékének. Ilyen módon a `print()` parancsot ritkábban hasznájuk, mint más programnyelvek esetén.

```
print(x)
[1] 0.5555556

set.seed(2004)
x <- runif(10)
dim(x) <- c(2,5)
x
           [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.11344774 0.2823110 0.5699885 0.3389075 0.5026153
[2,] 0.04885379 0.5672028 0.6407288 0.3655002 0.8683435

print(x, digits=2, justify = "right")
     [,1] [,2] [,3] [,4] [,5]
[1,] 0.113 0.28 0.57 0.34 0.50
[2,] 0.049 0.57 0.64 0.37 0.87
```

Szintén az eredmények kiíratására szolgál a `cat()` parancs, ami szintaxisát tekintve a `C/C++` programozási nyelv `printf()` függvényéhez hasonlít. (A `C/C++`-ban "mindent" függvénynek szokás nevezni.) A `formatC()` parancs szintén használható `C/C++` stílusú formattálására az eredményeknek.

Az aktuális nyomtatási beállításokat a tizedesjegyek számát illetően a `getOption("digits")` parancs segítségével kérdezhetjük meg. A kinyomtatandó tizedek számát a

```
options(digits=n)
```

parancs segítségével módosíthatjuk, ahol `n` egy egész szám `1` és `22` között. Az alábbi példa szemlélteti a parancsk használatát:

```
getOption("digits")
[1] 7
options(digits = 3)
getOption("digits")
[1] 3
```

Egy változó értékét igen egyszerűen kiírathatjuk a képernyőre. Ehhez csak a változó nevét kell begépelnünk és megnyomnunk az ENTER billentyűt. Használhatjuk a `print()` parancsot is, de erre nincs szükség, ha a parancssorból akarunk nyomtatni. Ha azonban saját függvényt definiálunk és ekkor szeretnénk nyomtatni, akkor ehhez használni kell a `print()` parancsot.

Néhány fontosabb jelölési konvenció R-ben

- Az R különbséget tesz a kis- és nagybetűk között, azaz az `"a"` és `"A"` különbözőek. Azaz `x` és `X` nem azonos:

  ```
  x <- 9
  X <- 32
  (x == X)
 [1] FALSE
  ```

- A változók és függvények nevei csak alfanumerikus karaktereket tartalmazhatnak (`A-Z`, `a-z`, `0-9`) valamint pontot `"."`. A nevek nem kezdődhetnek számjegyekkel. A pontokat (`"."`) használhatjuk a változónevekben határoló karakterként. Hasonlóképpen használhatjuk az aláhúzás jelet (`"_"`) az újabb (1.9 és újabb) R verziókban. Így a

  ```
  valtozo.1 <-5
  valtozo_1 <-7
  ```

  egyaránt érvényes változónekvek. A `"jaj-de-szep!"` nem érvényes változónév, mert alfanumerikus karakteren, ponton és aláhúzásjelen kívül más karaktert is tartalmaz. Próbáljuk ki!

  ```
  jaj-de-szep! <- 14
  ```

  Ez

  ```
  Error: syntax error
  ```

hibaüenetet eredményez. Egyrészt a "-" jelek miatt, másrész a "!" miatt. Szintén hibaüzenetet kapunk ha a JajDeSzep! változónevet használjuk. A

```
JajDeSzep <- 18
```

Értékadás működik, mert a változó nevében csak alfanumerikus karakterek szerepelnek.

- A parancsokat pontosvesszővel kell elválasztani egymástól vagy új sorba kell írni. Azaz

```
x <- 2
y <- 5
(x + y)
```

ugyanazt eredményezi mintha a parancsokat egyetlen sorba írjuk pontosvesszővel elválasztva:

```
x <- 2; y <- 5; (x + y)
```

- A csoportok elhatárolására a kapcsos zárójelek szolgálnak. A kerek zárójelekkel a műveletek végrehajtásának sorrendjét határozhatjuk meg. A szögletes zárójelekkel a vektorizált objektumok elemeit kérdezhetjük le. Például x[3] az x vektor harmadik elemét jelöli.
- Az R egy interpreteres nyelv, de be lehet építeni C/C++/Fortran kódokat is. Ez azonban messze meghaladja ennek a rövid ismertetőnek a kereteit.
- A "#" jellel kezdődő sorokat az interpreter figyelmen kívül hagyja, azaz ezek tartalmazzák a programkódra vonatkozó kommentárokat.
- Ha egy programsor olyan hosszú, hogy túlnyúlik a sor hosszán, akkor a következő sor elején ezt egy "+" jelű kurzor jelzi.
- Ha valamelyik parancssort újra szeretnénk futtatni, akkor ezeket nem kell újra begépelni, hanem a lefelé és felfelé mutató nyilak segítségével újra előhívhatjuk őket.

1.3. Adatstruktúrák R-ben

A modern programozási nyelvekhez hasonlóan az R egy *objektumorientált* nyelv. Az R-ben különösen hasznos az az álláspont, hogy minden objektum, az adatvektoroktól a bonyolult adatstruktúrákig és a függvényekig.

A vektor a legegyszerűbb adatstruktúra az R-ben és gyakorlatilag statisztikai szempontból is ez az alapvető adatfeldolgozási egység.

A vektor nem az egyetlen adatstruktúra az R-ben. Használhatunk mátrixokat és tömböket is. A tömbök többdimenziós mátrixok. A mátrixok és a tömbök a vektorokhoz hasonlóan ugyanolyan típusú (mode-ú) elemeket tartalmaznak. Az olyan esetekben amikor eltérő típusú adatokat szeretnénk összekapcsolni egy

adatstuktúrába, akkor R-ben *listákat* (*lists*) kell használni. A listák elemei lehetnek vektorok, mátrixok, vagy akár további listák is. R-ben a listák használatosak akkor is, ha egy függvény többféle értéket ad ereményként; a függvény eredményét ilyen formában kapjuk. A listák egy speciális típusát képviselik az *adatkeretek* (*data frame*) is, amelyeket szintén adathalmazok megadására használhatunk. Az adatkeretek numerikus és karakter adatokat tartalmazhat, amelyeket kiegészítünk a változók neveivel.

Vektorok

A vektoroknak a `c()` parancs segítségével adhatunk értéket:

```
x <- c(4,3,8,9)
x
[1] 4 3 8 9
```

Ha az `x` vektort további két elemmel, az `5`-el és a `2`-vel szeretnénk akkor ezt az alábbi módon érhetjük el:

```
c(x,5,2)
[1] 4 3 8 9 5 2
```

Ha karaktervektort készítünk, akkor a `c()` parancsot olyan módon kell használni, hogy karaktereket idézőjelek közé írjuk.

```
ord <- c("egy","kettő","három")
ord [1]
"egy"  "kettő"  "három"
```

A vektorok hosszát megtudhatjuk a `length(x)` parancs révén.

```
x <- c(4,3,8,9)
length(x)
[1] 4
```

Sorozatok készítése R-ben

Az R-ben igen kényelmes és sokoldalú parancsok állnak a rendelkezésünkre sorozatok generálására. A `seq()` parancsot használhatjuk sorozatok készítésére. A `rep()` parancs segítségével pedig ismétlődő elemeket tartalmazó sorozatokat készíthetünk. A sorozatok generálására a legegyszerűbb a `":"`, azaz a kettőspont művelet. A kettőspont műveletnek a vektorizáltság révén igen fontos szerepe van a vektorok és mátrixok elemeinek vagy az elemek egy részhalmazának megadásában. Egy vektort, amelynek elemei az az egész számok `1`-től `5`-ig, az alábbi módon generálhatjuk

```
x <- c(1:5)
x
```

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

Sőt, a fentieket még tovább rövidíthetjük

```
x <- 1:5
```

formában. Ha általánosabb sorozatokat szeretnénk előállítani, akkor a `seq()` és a `rep()` parancsokat használhatjuk. Például a

```
seq(from=1, to=4, by=0.6)
[1] 1.0 1.6 2.2 2.8 3.4 4.0
```

parancs egy olyan vektort generál, ami számokat tartalmaz `1`-től `4`-ig `0.6`-os lépésközzel. A fenti parancsot némileg tömörebben az alábbi formában is megadhatjuk:

```
seq(1, 4, 0.6)
```

A rövidebb változatot használva az egyes értékek poziciója rögzített a parancs argumentumában. Amennyiben a hosszabb formát választjuk, akkor tetszőleges sorrendben megadhatjuk a parancs paramétereit:

```
seq(to=4, from=1, by=0.6)
```

Ez a parancs az előzővel azonos eredményt ad. Ugyanez a parancs az argumentumban szereplő értékek nevének megadása nélkül hibaüzenetet eredményez:

```
seq(4, 1, 0.6)
Error in seq.default(4, 1, 0.6) : Wrong sign in 'by' argument
```

A `rep()` paranccsal bizonyos vektorokat ismételhetünk meg egy újabb vektorban annyiszor, ahányszor kívánjuk. Például a

```
rep(c(0,1,2),3)
```

parancs a `(0,1,2)` vektort megismétli háromszor és így a `(0,1,2, 0,1,2, 0,1,2)` vektort eredményezi. Kiadhatunk bonyolultabb parancsokat is. A

```
rep(c("a","b","c"), c(2,3,1))
```

parancs hatására az `("a","b","c")` vektor a `(2,3,1)` vektor szerint többszöröződik, olyan módon, hogy az első vektor elemei annyiszor fognak ismétlődni, amekkora szám a második vektor ugyanezen poziciójában szerepel. Azaz, az `"a"` egyszer fordul elő, `"b"` kétszer, és `"c"` egyszer fordul elő. Hasolóképpen működik a

```
rep(c(1,7,0), c(2,5,3))
```

is:

```
rep(c("a","b","c"), c(2,3,1))
[1] "a" "a" "b" "b" "b" "c"
```

és

```
rep(c(1,7,0), c(2,5,3))
[1] 1 1 7 7 7 7 7 0 0 0
```

A `seq()` és a `rep()` parancsok kombinálhatók is, amint azt az alábbi példa szemlélteti:

```
rep(seq(0,4,2), c(1,3,5))
[1] 0 2 2 4 4 4 4 4
```

A `seq(0,4,2)` paranccsal egy sorozatot állítunk elő `0`-tól `4`-ig, `2`-es lépésközzel. Azaz a sorozat `0 2 4` alakú. A `c(1,3,5)` azt jelenti, hogy ennek a sorozatnak az első elemét egyzser, a második elemét háromszor és a harmadik elemét ötször ismételtetjük meg. Ugyanezt az eredményt érjük el az alábbi paranccsal is:

```
rep(seq(0,4,2), seq(1,5,2))
[1] 0 2 2 4 4 4 4 4
```

Lényeges különbség a `seq()` és a `":"` valamint a `rep()` parancsok között, hogy a `rep()` parancs karakterek és logikai típusú változók esetén is használható, míg a `seq()` és a `":"` csak numerikus változók esetén használható.

A vektorizáltság a programozást jelentős mértékben egyszerűsíti és természetesebbé is teszi. Ha például két vektort szeretnénk összeadni, akkor nincs ehhez szükség ciklusokra, hanem megtehetjük ezt egyszerűen az `"x+y"` parancs révén, ahol `x` és `y` azonos dimeniójú vektorok. A vektorizáltság révén adódó funkcionalizmus és a kód tömörsége szokatlan lehet azok számára, akik már hozzászoktak a hagyományos programozási nyelvek logikájához.

A vektorok elemeinek ugyanolyan típusúnak kell lenniük. A típusokat `mode`-nak nevezik az R-ben. Az R-ben rendelkezésre álló típusok (`mode`) az alábbiak:

- `character` – stringek, azaz karakterek sorozatának,
- `complex` – komplex számoknak,
- `logical` – logikai változóknak (`TRUE` és `FALSE`), és
- `numeric` – valós számok tárolására szolgál.

Vektorok indexelése

A vektorok indexeinek jelölésére a szögletes zárójelek (`"[ ]"`) szolgálnak. Például `x[2]` jelöli az `x` vektor második elemét. Az R-ben a józan észnek megfelelően a vektorok első elemét `1`-el indexeljük és nem `0`-val, mint az "igazi programozóknak" készült programnyelvekben. A kettőspont operátorral (`":"`) a vektor egy részét kiválaszthatjuk; például `x[2:7]` az `x` vektor elemit jelenti a másodiktól a hetedikig:

```
x <- 1:10
x[2:7]
```

Szintén használhatunk logikai indexeket is. Az `x[x > 3]` jelölés a háromnál nagyobb elemeket jelöli:

```
x[x > 3]
[1]  4  5  6  7  8  9 10
```

Ha az `x` vektor ötödik elemének értékét szeretnénk megtudni, akkor ez olyan módon kaphatjuk meg, hogy begépeljük a parancssorba, hogy `x[5]`, aminek hatására kiíródik a vektor ötödik elemének értéke. Ha nem a vektor egyetlen elemének az értékére vagyunk kyváncsiak, hanem az értékek egy összefüggő tartományára, akkor ezt a `":"` parancs segítségével kaphatjuk meg. Például az alábbi példában a 10 elemű vektor esetén a harmadiktól a hatodik eleméig iratjuk ki az értékeket:

```
x <- c(1,2,3,4,7,8,9,10)
x[3:6]
[1] 2 3 4 7
```

Nem szükségszerű azonban, hogy az indexek összefüggő tartományt alkossanak. Tetszőleges elemek is lekérdezhetők. Az első, ötödik és hetedik elem értékét az alábbi módon irathatjuk ki:

```
x[c(1,5,7)]
[1] 1 5 7
```

Az R-ben igen kényelmesen lehet elhagyni a vektorizált objektumok elemeit. A `":"` parancs segítségével törölhetünk értékeket egy vektorból. Ha az `x` vektor elemei közül a negyediktől a hatodikig terjedőket el kívánjuk hagyni, akkor ezt olyan módon tehetjük meg, hogy negatív előjellel adjuk meg ezt a tartományt:

```
x[-(2:4)]
[1] 0 4 7 8 9 10
```

Ha `x` egy numerikus vektor, mint az eddigi példákban, akkor `x[x < 5]` egy olyan vektort eredményez, aminek az elemei kisebben `5`-nél. Ez úgy lehetséges, hogy `"x < 5"` egy logikai vektor: `x` minden elemére `"< 5"` kiértékelődik `TRUE`-ként vagy `FALSE`-ként:

```
x
[1] 1 2 3 4 7 8 9 10


x < 5
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE
```

Az `x[x < 5]` vektor elemei azok lesznek, amelyek esetén `x < 5` értéke `TRUE`:

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

A logikai indexek egészen bonyolultak is lehetnek, amint azt az alábbi példa is mutatja:

```
x[(x >= 4) & (x <= 7) & (x != 5)]
[1] 4 6 7
```

Mátrixok

A mátrixok (`matrix`) olyan kétdimenziós adatstruktúrák, amelyek `nrow` számú sorból és `ncol` számú oszlopból állnak.

Az `xm` mátrixot, ami két sorból és három oszlopból áll, az alábbi módon definiálhatjuk

```
xm <- matrix(c(1, 2, 1, 2, 1, 2), nrow = 2, ncol = 3, dimnames = NULL)
```

ahol az `nrow` paranccsal adhatjuk meg a sorok számát és az `ncol` paranccsal az oszlopok számát. A `dimnames` paranccsal "nevet" adhatunk a soroknak és az oszlopoknak. Ha a `dimnames = NULL` parancsot használjuk, akkor a soroknak és az oszlopoknak nincs külön nevük, csak a sorszámuk. A mátrix elemeit a `copy`, azaz `c()` paranccsal adhatjuk meg, ahol az egyes értékeket vesszővel választjuk el egymástól. Ha beírjuk a mátrix nevét, akkor megnézhetjük az így létrehozott adatstruktúrát:

```
>  xm
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

Sorfolytonosan is megadhatjuk a mátrix elemeit. Ekkor az argumentumban meg kell adnunk a `byrow = FALSE` opciót:

```
xm <- matrix(c(1, 1, 1, 2, 2, 2), nrow = 2, ncol = 3,
             byrow = FALSE, dimnames = NULL)
```

Számos más módon is létrehozhatunk mátrixokat. Például olyan módon, hogy azonos hosszúságú vektorokat sorvektorokként vagy oszlopvektorokként "összeragasztjuk", összekapcsoljuk. Legyen

```
xr1 <- c(1, 1, 1)
xr2 <- c(2, 2, 2)
```

és

```
xm <- rbind(xr1, xr2)
```

szolgál az `xv1` és `xv2` parancsok összekapcsolására. Ekkor a mátrix sorainak neve a sorvektorok nevével lesz azonos:

```
> xm
    [,1] [,2] [,3]
xv1    1    1    1
xv2    2    2    2
```

Ha oszlopvektorokat szeretnénk egy mátrixba összekapcsolni, akkor a `cbind()` parancsot használjuk:

```
xc1 <- c(1, 2)
xc2 <- c(2, 2)
xc3 <- c(1, 2)

xm <- cbind(xc1, xc2, xc3)
```

Ekkor

```
> xm
     xc1 xc2 xc3
[1,]   1   2   1
[2,]   2   2   2
```

és a mátrix oszlopainak neve a vektorok nevével fog megegyezni.

A mátrix sorainak és oszlopainak nevét az alábbi módon adhatjuk meg

```
xm <- matrix(c(1,2, 3,4), nrow = 2, ncol=2, byrow=TRUE,
      dimnames = list(c("sor1", "sor2"), c("oszlop1", "oszlop2")))
```

Ekkor a mátrix

```
> xm
     oszlop1 oszlop2
sor1       1       2
sor2       3       4
```

formájú lesz.

1.4. A programozás elemei R programnyelven

Mivel az R vektorizált, így a ciklus utasítások használatára ritkán van szükség. Ha mégis szükség van rá, hogy explicit módon használjunk ciklusképző utasításokat, akkor az egyéb programozási nyelvek esetében szokásos `for`, `while` és `repeat` utasítások a rendelkezésünkre állnak. A leggyakrabban a `for` utasítás használatos. Ezt ismertetjük részletesebben. A programozási feladatok rendszerint mindhárom ciklus segítségével megfogalmazhatók csak a feladat jellegétől függően az egyik vagy a másik ciklus használata természetesebb és egyszerűbb.

Az R egy vektorizált programozási nyelv, hasonlóan az S/S++-hoz, a MATLAB-hoz és az Octave-hoz. Ez rendkívül hatékonnyá teszi a programozási nyelvet, ami bőven ellensúlyozza a kezdeti szokatlanságát a vektorizált programozásnak. A vektorizált azt jelenti, hogy a vektorokon és mátrixokon elvégzett műveleteket nem kell a vektor vagy mátrix minden elemére felírni, hanem elegendő a vektor vagy mátrix egészére. Például, ha `xv` egy vektor, akkor az `xv` vektor elemeinek összege a `sum(xv)` paramccsal definiálható. Legyenek az `xv` vektor

elemei az egész számok egytől ötig. Ekkor a vektor elemeinek összege éppen 15. R-ben ez következőképpen (és számos egyéb módon is) adható meg:

```
xv <- c(1, 2, 3, 4, 5)
sum(xv)
[1] 15
```

A saját függvények definiálása során a ciklusképző utasításokat és az `if-else` feltételes elágazást programozó utasítást használhatjuk programozási struktúrákként. Ezeket ismertetem röviden, hogy a könyv további részében lévő egyszerű alkalmazások során a használatuk nyomon követhető legyen.

Saját függvények definiálása

Könnyen definiálhatunk saját függvényeket is R-ben. Ehhez egy egyszerű editor áll a rendelkezésünkre, amit a `fix()` paranccsal indíthatunk el. A zárójelben kell megadnunk a függvény nevét. A

```
fix(megHat)
```

parancs begépelésével az editor elindítása után egy nagyobb nevű saját függvényt definiálhatunk. Az editor elinduláskor megadja a függvények definiálásához szükséges minimális szintaxist:

```
function()
{
}
```

Ezután gépeljük be a programkód további részét:

```
> megHat
function (x)
{
  x <- x + 6
  x
}
```

Amikor a programkód begépelésével elkészültünk, akkor válasszuk a legödülő menüből a "*File*", majd az "*Exit*" opciót. Majd próbáljuk ki a függvényt, ami mindössze megnöveli kettővel a függvény argumentumában szereplő szám értékét:

```
megHat(3)
[1] 9
```

A beépített editornál sokkal kényelmesebb a szokásos szövegszerkesztőnket használni és ebben írni a programkódot, majd az operációs rendszer asztalának (*desktop*) memóráján keresztül átmásolni az R-be.

`for` ciklus

Mivel az R vektorizált programozási nyelv, így a ciklusok, és így a `for` ciklus is, sokkal kevesebb szerepet kap, mint a hagyományos programozási nyelvek esetében. Jóval gyorsabb a számolás, ha ugyanazt a számolási feladatot vektorizált formában kódoljuk, mint valamilyen ciklust használva. Igen ritkán adódik olyan számolás, ami nem oldható meg vektorizáltan a tapasztalt R programozó számára.

A `for` ciklus szintaxisa az alábbi

```
for ( <index változó> in <index vektor> ) {
   ciklus mag
}
```

Az `<index változó>` szolgál számlálóként és az `<index vektor>` tartalmazza az értékeket, amit a számláló befut. Ha `1`-től `100`-ig össze szeretnénk adni a számokat a `for` ciklus használatával, akkor ez a következő módon történhet:

```
total <- 0
for ( i in 1:100 ) {
 total <- total + i
} total
```

Ugyanez némileg egyszerűbben, vektorizálva felírható az alábbi módon:

```
sum(1:100)
[1] 5050
```

Az indexvektornak nem feltétlenül kell összefüggő halmaznak lennie; megadhatunk tetszőleges vektort is, amint azt az alábbi példa mutatja:

```
for (i in c(2,7,18)) {
  print(i/2)
}
[1] 1
[1] 3.5
[1] 9
```

Az indexvektor bármilyen típusú vektor lehet:

```
for (size in c("kicsi", "közepes", "nagy")) {
 print(size)
}
```

A `while` és a `repeat` ciklusok

A `while` utasítás esetén a ciklusfeltételt kiértékeli a program és `TRUE` (`igaz`) érték esetén hajtódik végre a program:

```
while ( <feltétel> ) { számolandók; }
```

A `while` ciklus esetén nem kell megadni, hogy hányszor kell a számolandókat végrehajtani, mert mindaddig folyik a számolás, míg a feltétel értéke `FALSE` (`hamis`) nem lesz. Egy triviális példa a ciklus használatára:

```
i <- 4
while ( i < 7 ) {
  i <- i + 1
  print( i )
}
[1] 5
[1] 6
[1] 7
```

A `repeat` ciklus igen hasonló a `while` ciklushoz; ekkor azonban egyszer mindenképpen végrahjtódnak a számolási utasítások és csak utána teszteli a program a feltételeket. Ha a feltétel értéke `TRUE` (`igaz`), akkor a program kilép a ciklusból.

```
repeat {
  számolandók;
  if ( <feltétel> ) break;
}
```

A `repeat` ciklus használatát szemlélteti az alábbi egyszerű példa:

```
i <- 0
repeat {
  i <- i + 1;
  print( i );
  if ( i >= 3 ) break;
}
[1] 1
[1] 2
[1] 3
```

Az `if-else` utasítás

Feltételes utasítás az `if-else` parancs az alábbi formában használható, ha egyetlen sorba írjuk

```
if ( kifejezés ) utasítás else utasítás
```

és az alábbi módon, ha az `if` vagy az `else` ágon több utasítást szeretnénk végrehajtani:

```
if ( kifejezés ) {
  utasítások
} else {
```

```
  utasítások
}
```

Az interpreter megvisgálja a kifejezés értékét; ha ez `TRUE`, akkor az `if` ágon lévő utasítást hajtja végre, egyébként pedig az `else` ágon lévőt. Az alábbi példa szemlélteti az `if-else` utasítás használatát:

```
x <- 5;  y <- 17
if ((x-y) > 0) {
  print("A különbség pozitív.")
} else {
  print("A különbség negatív.")
}
```

Ha nincs `else` ág és a kifejezés értéke `FALSE`, akkor nem hajtódik végre utasítás, azaz nem történik semmi. Ezt szemlélteti az alábbi egyszerű példa:

```
a <- FALSE
if (a) print(a)
```

Ilyen esetben a teljes kifejezés értéke `NULL`. Erről meggyőződhetünk, ha az `if` elágazó utasítás eredményét átirányítjuk egy `xL` logikai változóba és kiíratjuk az `xL` értékét, amint azt az alábbi példa mutatja:

```
a <- FALSE
xL <- if (a) print(a)
xL
NULL
```

1.5. Segítségkérés R-ben

Többféleképpen is kérhetünk segítséget R-ben. Ha tudjuk a feladat elvégzéséhez szükséges parancs nevét és csak a részleteket szeretnénk pontosan tudni, akkor be kell gépelnünk a parancs nevét egy kérdőjel után. Például, ha tudjuk, hogy átlagot a `mean` parancs segítségével számolhatunk, de nem tudjuk a további részleteket, akkor

```
?mean
```

begépelésével az alábbi információt kapjuk. Szerencsére az információ angolul van, így még nyelvtudásunkat is gyakorolhatjuk.

```
mean                    package:base                    R Documentation

Arithmetic Mean

Description:
    Generic function for the (trimmed) arithmetic mean.
```

```
Usage:
    mean(x, ...)

    ## Default S3 method:
    mean(x, trim = 0, na.rm = FALSE, ...)

Arguments:

      x: An R object.  Currently there are methods for numeric data
         frames, numeric vectors and dates.  A complex vector is
         allowed for 'trim = 0', only.

   trim: the fraction (0 to 0.5) of observations to be trimmed from
         each end of 'x' before the mean is computed.

  na.rm: a logical value indicating whether 'NA' values should be
         stripped before the computation proceeds.

    ...: further arguments passed to or from other methods.

Value:

    For a data frame, a named vector with the appropriate method being
    applied column by column.

    If 'trim' is zero (the default), the arithmetic mean of the values
    in 'x' is computed.

    If 'trim' is non-zero, a symmetrically trimmed mean is computed
    with a fraction of 'trim' observations deleted from each end
    before the mean is computed.

References:

    Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
    Language_. Wadsworth & Brooks/Cole.

See Also:
    'weighted.mean', 'mean.POSIXct'

Examples:
    x <- c(0:10, 50)
    xm <- mean(x)
    c(xm, mean(x, trim = 0.10))
```

```
data(USArrests)
mean(USArrests, trim = 0.2)
```

Ugyanezt a segítséget megkaphatjuk, ha a

```
help(mean)
```

parancsot gépeljük be.

Ha valamilyen speciális karakter használatáról szeretnénk információt kapni, akkor ezt a karaktert vagy karakterek sorozatát idézőjelben kell beírni; például

```
help("<-") vagy ? "<-"
```

begépelése az alábbi információval szolgál:

```
assignOps              package:base              R Documentation

Assignment Operators

Description:
     Assign a value to a name.

Usage:
     x <- value
     x <<- value
     value -> x
     value ->> x

     x = value

Arguments:
       x: a variable name (possibly quoted).
   value: a value to be assigned to 'x'.

Details:

There are three different assignment operators: two of them have
leftwards and rightwards forms.

The operators '<-' and '=' assign into the environment in which
they are evaluated.  The '<-' can be used anywhere, but the '=' is
only allowed at the top level (that is, in the complete expression
typed by the user) or as one of the subexpressions in a braced
list of expressions.
```

```
The operators '<<-' and '->>' cause a search to made through the
environment for an existing definition of the variable being
assigned. If such a variable is found then its value is redefined,
otherwise assignment takes place globally.  Note that their
semantics differ from that in the S language, but are useful in
conjunction with the scoping rules of R.

In all the assignment operator expressions, 'x' can be a name or
an expression defining a part of an object to be replaced (e.g.,
'z[[1]]').  The name does not need to be quoted, though it can be.

The leftwards forms of assignment '<- = <<-' group right to left,
the other from left to right.

Value: 'value'.  Thus one can use 'a <- b <- c <- 6'.

See Also: 'assign', 'environment'.
```

Adódhat olyan helyzet, hogy segítségre van szükség, de nem tudjuk a pontos parancsot. Ilyen helyzetekben is igen hatékony segítséget nyújt az R A `help.search()` és az `apropos()` parancsok használhatók kulcsszavak keresésére. A keresés során használt kulcsszavakat idézőjelek közé kell írni.

```
help.search("skewness")
```

segítségkérés az alábbi eredményt adja, jóllehet az eredmény függ attól is, hogy milyen kiegészítő programcsomagok ("*library*") vannak installálva.

```
Help files with alias or concept or title matching 'skewness'
using fuzzy matching:

k3.linear(boot) Linear Skewness Estimate
davies.moment(Davies) Moments of the Davies distribution
skewness(e1071) Skewness basicStatistics(fBasics)
Basic Statistics Summary

Type 'help(FOO, package=PKG)' to inspect entry 'FOO(PKG) TITLE'.
```

Ha nem vagyunk kíváncsiak a "`help`" teljes szövegére, akkor az alábbi módon kérhetjük, hogy az adott parancs használatára mutasson be egy példát vagy példákat; az átlag számolására szolgáló `mean` parancs használatának bemutatását kérjük:

```
example(mean)
```

Az eredmény az alábbi példa:

```
> example(mean)
```

```
mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.1))
[1] 8.75 5.50
mean> data(USArrests)
mean> mean(USArrests, trim = 0.2)
Murder  Assault UrbanPop     Rape
  7.42   167.60    66.20    20.16
> x
[1]  0  1  2  3  4  5  6  7  8  9 10 50
```

Ha az `example()` paranccsal kérünk információt valamelyik statisztikai vagy egyéb függvény használatáról, akkor ezt a fenti formában kapjuk, ahol a "`>`" kurzor előtt minden példaként szolgáló parancs esetén kiírja a rendszer a függvény nevét, jelen esetben az, hogy "`mean`".