Informatikai alapok, R változók

Tamas Kadlecsik

February 14, 2016

Jelmagyarázat

Szakkifejezés, érdemes megjegyezni, a jegyzet végén található egy szótár. A kifejezések angol megfelelőjét van csak igazán értelme megjegyezni, ugyanis válaszokat csak angolul fogunk találni a problémáinkra. Vizsgán pedig mindent lehet használni. Nagy Google-t is.

Masszív elmélet, nem feltétlenül érthető elsőre (=nekem anno sokáig tartott megérteni). Nem árt, ha tudjátok, kis gyakorlattal majd valószínűleg a helyére kattan a dolog.

Konzol magyarázat

Ha bármilyen matematikai (logikai, vagy aritmetikai) **kifejezés**t írunk a **konzolba** és entert ütünk, a kifejezés értéke megjelenik a konzolon:

```
1+2
```

```
## [1] 3
```

```
2/3
```

```
## [1] 0.6666667
```

```
1*2
```

```
## [1] 2
```

```
1-6
```

```
## [1] -5
```

Amennyiben egy adott értéket később is el szeretnénk érni, változóhoz kell rendelnünk. Ezt a **<- értékadó operátorral** tehetjük meg.

```
amy <- 20
bob <- 22
charlie <- 19
donna <-26
amy
```

```
## [1] 20
```

A kifejezés ugyanúgy **kiértékelődik** a változó mindenkori értéke alapján

```
amy + charlie
```

```
## [1] 39
```

```
s <- amy + charlie
```

Ha pusztán a változó nevét írjuk a konzolba, és nem végzünk rajta műveletet, visszakapjuk a változó értékét.

```
s
```

```
## [1] 39
```

```
s <- amy+charlie+donna+bob
s
```

```
## [1] 87
```

Több értéket összefűzhetünk egy több elemű **vektorrá** a `c()` (**concatenate**) függvénnyel. A függvénynek átadott **argumentumokat** (jelen esetben amy, Charlie, bob és donna) vesszővel elválasztva adjuk meg A függvények hívása mindig a `([args])` **függvényhívó operátorral** történik. Vannak függvények amelyeknek nincs szükségük argumentumra a működéshez, lásd:

```
q()
getwd()
```

Az argumentumok mindig azokat a változókat jelentik, amelyen a függvény végrehajtódik. Például `c()` függvény esetbén azon váltózókat, vagy értékeket adjuk át argumentumként, amelyeket vektorrá szeretnénk fűzni. `sum()` esetén ez/ezek azon érték/értékek lesz/nek amelyeknek összegét szeretnénk megkapni.

```
ages <- c(amy, charlie, bob, donna)
ages
```

```
## [1] 20 19 22 26
```

A `sum` függvénnyel egy vagy több vektor értékeinek összegét kaphatjuk.

```
sum(ages)
```

```
## [1] 87
```

A `mean` függvénnyel egy vagy több vektor értékeinek átlagát kaphatjuk.

```
mean(ages)
```

```
## [1] 21.75
```

A `median` függvénnyel egy vagy több vektor értékeinek mediánját kaphatjuk.

```
median(ages)
```

```
## [1] 21
```

Minden függvényről kaphatunk leírást, és magyarázatot, ha a neve elé `?`-t írunk, majd entert ütünk. Itt a "Description" résznél a függvény általános leírását kapjuk, utána a lehetséges argumentumokat. A függvények leírásának pontosabb értelmezéséről a függvényekről szóló órán less szó.

```
?mean
```

A vektorok adott sorszámú elemét `[index]`-ként tudjuk elérni

```
ages[1]
```

```
## [1] 20
```

```
ages[2]
```

```
## [1] 19
```

A változók típusát fejben tartan minden programnyelvben fontos! Bizonyos függvények csak adott típusú változókon értelmezettek. Léteznek **atomi** (**primitív**), és **összetett** típusok. Az atomi változók lehetnek:

- **Skalár / Szám**:
  - **Integer / Egész szám**: Ha feltétlenül integer típusú számot szeretnénk kapni `L` betűt kell írnunk a számok után. Pl: `1L`, `25L` Ezesetben tört értékeket nem adhatunk meg. Ritkán van rá szükség
  - **Double (Float, Real) / Valós szám (lebegőpontos szám)**: Az alapértelmezett számábrázolás. Ha bármilyen számot leírunk annak típusa alapértelmezésben „double"
  - **Complex / Komplex szám**: magasabb matematikai dolog, ld. Wikipédia ;))
- **Character (String) / Karakter (karakterlánc, szöveg)**: A változóktól eltérően a karakter típusú értékeket mindig "-ek, vagy '-ek közé kell tenni.
- **Logical / Logika**: Értékük igaz vagy hamis lehet. Ezt `T`/`F`, vagy `TRUE` és `FALSE`-ként jelöljük. Leggyakrabban kapcsolóként funkcionálnak.

Az atomi változók pedig összetett típusokba rendezhetők, eddig beszétünk a vektorról. R-ben a vektor is atomi típusnak számít, mivel csak egyféle típusú változó kerülhet bele, és az alapműveletk végrehajtásakor is egy egységnek tekinti az interpreter. Sőt, ha csak egyetlen értéket rendelünk egy változóhoz az automatikusan egy egyelemű vektor lesz Tehát, amikor azt mondjuk, hogy egy vektor „double", egy másik pedig „character" típusú valójában arról beszélünk, hogy minden egyes értéke „double", a másiknak pedig minden egyes értéke „character" típusú. A továbbiakban beszélünk még mátrixokról, listákról és táblázatokról (dataframe).

Egy változó típusát a typeof(változó) függvénnyel tudjuk lekérni:

```
typeof(ages)
```

```
## [1] "double"
```

```r
logic <- c(T, F, TRUE, FALSE)
logic
```

```
## [1]  TRUE FALSE  TRUE FALSE
```

```r
typeof(logic)
```

```
## [1] "logical"
```

```r
word <- c('apple', 'banana')
typeof(word)
```

```
## [1] "character"
```

Amennyiben egy szöveget nem teszünk idézőjelek, vagy aposztrófok közé, akkor változónak értelmezi! Jó esetben még nem létezik a változó, és a script elhasal, rossz esetben lefut, és nem azt az eredményt kapjuk, amire szükségünk lenne.

```r
# w <- c(apple, banana)
# Error: object 'apple' not found
```

Ha bárhova #-karaktert írunk onnantól az adot sor nem fut le. Ezt nevezik commentnek. Kíválóan alkalmas arra, hogy a kódunkban megmagyatázzuk a nem teljesen érthető dolgokat. Itt most azért csak kommentel írtuk az értékadást, mert az apple és a banana változók még nincsenek definiálva, így a második sorba írt hibát kapnánk. Mivel a jegyzetet kódba írom, ezért ez a hívás nekem se fut le, így nem is tudom legenerálni a szöveget.

```r
words <- c('apple', 'banana')
words
```

```
## [1] "apple"  "banana"
```

```r
apple <- 1
banana <- 2
fruits <- c(apple, banana)
fruits
```

```
## [1] 1 2
```

Ez már lefut. Mint már említettem, skalár vektorok esetén a matematikai műveletek a változó összes értékén végrehajtódnak!

```r
ages + 1
```

```
## [1] 21 20 23 27
```

```
ages
```

```
## [1] 20 19 22 26
```

Amennyiben egy változó értékét szeretnénk módosítani, a változót **fölül kell írnunk**! Ezt úgy tudjuk megtenni, hogy a változónk végrehajtunk egy műveletet, majd az utasítás értékét hozzárendeljük az eredeti változóhoz. Ebből látszik, hogy mindig a <- operátor jobb oldala értékelődik ki először, majd a kiértékeléskor kapott érték rendelődik a bal oldalon található változóhoz.

```
ages <- ages + 1
ages
```

```
## [1] 21 20 23 27
```

Eddig egydimenziós vektorokról beszéltünk. Ezeket két- vagy többdimenziós mátrixokká fűzhetjük össze, azonban ritkán van szükségünk 2D-nél többre, így ezt most nem is tárgyaljuk. Minden, amit eddig elmondtunk a vektorokról igaz a mátrixokra is (ld. feljebb). A sum(mátrix) függvény a mátrix ÖSSZES elemének összegét adja például.

A vektorok mátrixokká való összefűzését az rbind() (row bind, a vektorok a mátrix sorai lesznek), és cbind() (column bind, a vektorok a mátrix oszlopai lesznek) függvényekkel végezhetjük. Az argumentumok sorrendje fontos, hiszen az összefűzött értékek a megadott sorrend szerint fűződnek össze, melyekre ezek után leggyakrabban sorszámukkal (indexükkel) fogunk hivatkozni.

A cbind, és rbind függvényekkel természetesen mátrixokhoz is csatolhatunk vektorokat, sőt, mátrixokat is összefűzhetünk. Azonban figyelnünk kell, hogy cbind esetén a két mátrix ugyanannyi oszlopból, míg rbind esetén ugyanannyi sorból álljon. Itt is igaz, hogy a művelet során új mátrix jön létre, melyet változóba kell mentenünk!

```
height <- c(192, 165, 202, 185)
weight <- c(65, 102, 85, 78)
m1 <- rbind(ages, height, weight)
m2 <- cbind(ages, height, weight)
```

A létrejövő mátrixok oszlopai cbind esetén megkapják az összefűzött változók nevét. rbind esetén ez a sorokra igaz.

```
m2
```

```
##      ages height weight
## [1,]   21    192     65
## [2,]   20    165    102
## [3,]   23    202     85
## [4,]   27    185     78
```

A mátrix egy adott elemére matrix[sor, oszlop]-ként tudunk hivatkozni. Egész sorra matrix[sor,], egész oszlopra matrix[,oszlop]-ként. Figyeljünk a vesszők megfelelő használatára!

```
m2[1,]
```

```
##   ages height weight
##     21    192     65
```

```
m2[,1]
```

```
## [1] 21 20 23 27
```

```
m2[2,3]
```

```
## weight
##    102
```

Mint már említettük, a sum(mátrix) függvény a mátrix ÖSSZES elemének összegét adja. Ugyanez igaz az összes aggregáló függvényre (pl: mean(), median() stb.).

```
sum(m1)
```

```
## [1] 1165
```

```
sum(m2[,1])
```

```
## [1] 91
```

```
m1
```

```
##        [,1] [,2] [,3] [,4]
## ages     21   20   23   27
## height  192  165  202  185
## weight   65  102   85   78
```

```
m1[,1]
```

```
##   ages height weight
##     21    192     65
```

```
m2
```

```
##      ages height weight
## [1,]   21    192     65
## [2,]   20    165    102
## [3,]   23    202     85
## [4,]   27    185     78
```

```
m2[,1]
```

```
## [1] 21 20 23 27
```

```
sum(m2[,1])
```

```
## [1] 91
```

Lévén a mátrix is atomi értékekből kell álljon, a typeof() függvény a teljes mátrixra igaz típust fog visszaadni.

```
typeof(m1)
```

```
## [1] "double"
```

Az atomiság fényében, ha egy adott típusú atomi változóhoz más típusú váltózót füzünk (akár valamelyik bind-dal, akár c()-vel), implicit típuskonverzió történik. Vagyis az új változó típusa olyan lesz, hogy meg tudja tartani az összes addigi értéket. Lássuk ezt a gyakorlatban:

```
word <- c('apple', 'banana', 'kiwi', 'orange')
word
```

```
## [1] "apple"  "banana" "kiwi"   "orange"
```

```
typeof(word)
```

```
## [1] "character"
```

Eddig létrehoztunk egy karakter típusú változót, most hozzáfűzzük a double, vagyis skalár / szám típusú mátrixunkhoz.

```
m1 <- rbind(m1, word)
m1
```

```
##        [,1]    [,2]     [,3]   [,4]
## ages   "21"    "20"     "23"   "27"
## height "192"   "165"    "202"  "185"
## weight "65"    "102"    "85"   "78"
## word   "apple" "banana" "kiwi" "orange"
```

Minden szám idézőjelbe kerül, vagyis innentől ők is karakter típusúak. Találkozhatunk ezzel a hibával, ha magyar excelbe 12.1 jellegű számot írunk, vagy fordítva, angol excelbe 12,1-et (angolszászok, meg a hülye tizedespontjuk). Ránézve mi tudjuk, hogy a 12.1 egy szám csak nem úgy írva mint ahogy megszoktuk, a gép azonban nem, és szövegként értelmezi. Ugyanez történik R-ben is, csak itt több féle típusunk van.

A típuskonverzió a következő sorrendben történik.

logical < integer < double < complex < character

Vagyis logikai + integer típusból integer lesz, integer + double-ből double, ahogy logikai + doubleből, és logaikai + integer + double-ből is logikai, és így tovább. A különböző típusú változók összefűzését, és különböző típusú változókkal történő matematikai műveletek kezdők számár jó gyakorlás lehetnek!

Kezdésképpen: matematikai műveletet csak skalár változókkal tudunk végezni!^1 Ha karakterhez akarunk egyet adni hibát kapunk.

```
# m1 + 1
# Error in matrix1 + 1 : non-numeric argument to binary operator
```

Ismét csak komment mögé vagyok kénytelen bújtatni a híást

A ":" operátorról eddig nem esett szó. Gyakorlatilag –tól –igként funkcionál:

```
1:20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

És természetesen használhatjuk adott mátrix első 3 sorának lekérésére. (vagy a másodiktól az 5.-ig, ha éppen úgy tartja kedvünk. Ugyanez igaz vektor elemeire is.)

```
m1[1:3,]
```

```
##        [,1]  [,2]  [,3]  [,4]
## ages   "21"  "20"  "23"  "27"
## height "192" "165" "202" "185"
## weight "65"  "102" "85"  "78"
```

Mint látható `1:3 == c(1,2,3)`. Folytonos értékek generálásra kiváló, azonban az ages, height, weight változóinkat nem tudtuk volna vele létrehozni. Ahogy karaktereknél is teljesen hasztalan. Gyakorlásként próbáljuk ki, mit adna a matrix1[c(1,3)] kifejezés!

Explicit típuskonverzió történik, amikor az „as" függvényeket használjuk. Létezik as.numeric (mint numerikus), as.double, stb. Az összes primitív és összetett (as.vector) típusú változóhoz találunk konvertáló függvényt.

```
characters <- m1[c(1,2,3), ]
characters
```

```
##        [,1]  [,2]  [,3]  [,4]
## ages   "21"  "20"  "23"  "27"
## height "192" "165" "202" "185"
## weight "65"  "102" "85"  "78"
```

```
num.matrix <- as.numeric(characters)
num.matrix
```

```
##  [1]  21 192  65  20 165 102  23 202  85  27 185  78
```

```
typeof(characters)
```

```
## [1] "character"
```

```
typeof(num.matrix)
```

```
## [1] "double"
```

Különböző utsaításokat egybe is fűzhetünk és szét is szedhetünk:

```
first.row <- as.numeric(m1[1,])
first.row
```

```
## [1] 21 20 23 27
```

A fenti utasítás egyenértékű a következő két utasítással:

```
first.row.as.character <- m1[1,]
first.row <- as.numeric(first.row)

first.row.as.character
```

```
## [1] "21" "20" "23" "27"
```

```
first.row
```

```
## [1] 21 20 23 27
```

(Annak megértése, hogy mi is egy utasítás, lehet eltart egy darabig nem kell megijedni. Nem azonnal épül ki az összes új kapcsolat az agyban)

A mátrixok, és egyéb kétdimenziós változók sor, és oszlopneveit a rownames() és colnames() függvényekkel kérhetjük le.

```
colnames(m1)
```

```
## NULL
```

```
m1
```

```
##        [,1]    [,2]     [,3]   [,4]
## ages   "21"    "20"     "23"   "27"
## height "192"   "165"    "202"  "185"
## weight "65"    "102"    "85"   "78"
## word   "apple" "banana" "kiwi" "orange"
```

Amennyiben a colnames(), és rownames függvények visszatérési értékéhez értéket rendelünk, átírhatjuk az addigi sor- és oszlopneveket.

```
colnames(m1) <- c('amy', 'bob', 'charlie', 'donna')
m1
```

```
##        amy     bob      charlie donna
## ages   "21"    "20"     "23"    "27"
## height "192"   "165"    "202"   "185"
## weight "65"    "102"    "85"    "78"
## word   "apple" "banana" "kiwi"  "orange"
```

```
colnames(m2)
```

```
## [1] "ages"   "height" "weight"
```

A sor- és oszlopindexek helyett, adott esetben használhatjuk a sor- és oszlopneveket is. Ez kezdetben könnyebbséget jelenthet, de majd meglátjuk, miért nem túl „programozós”. Meg így amúgy is többet kell gépelni.

```
m1[,"amy"]
```

```
##    ages  height  weight    word
##    "21"   "192"    "65" "apple"
```

```
m1["ages","amy"]
```

```
## [1] "21"
```

A következő összetett típus amivel foglalkozunk a lista. Listákba bármilyen típusú változókat pakolhatunk, még listákat is. A következőben beteszünk egy egyelemű vektort, melynek értéke egy, második értékként beteszünk egy egyelemű vektort, melynek értéke 'apple', és beteszünk egy double típusú mátrixot.

```
my.list <- list(1, 'apple', m2)
```

Látható, hogy nem atomi típusról beszélünk:

```
typeof(my.list)
```

```
## [1] "list"
```

Azonban jelen listánk értékei atomi típusúak. Ez persze lehetne másképpen is. Lehetnének listák is. Azokban is lehetnének listák stb. de nem a végtelenségig! A legalján mindig kell legyen legalább egy atomi változó! A listák elemeit [[index]]-ként tudjuk elérni.

```
my.list[[1]]
```

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

```
typeof(my.list[[1]])
```

```
## [1] "double"
```

Jelen esetben a listánk 3. értéke a mátrix, amit beletettünk.

```
my.list[[3]]
```

```
##      ages height weight
## [1,]   21    192     65
## [2,]   20    165    102
## [3,]   23    202     85
## [4,]   27    185     78
```

`my.list[[3]] == m1`. Kimondva: a my.list 3. elemének értéke egyenlő m1 értékvel. Vagyis továbbra is, ha fölülírjuk matrix1-et az my.list[[3]]-on nem változtat semmit. Ez azt is jelenti, hogy `my.list[[3]][,"height"] == m1[,"height"]`.

```
my.list[[3]][,"height"]
```

```
## [1] 192 165 202 185
```

```
my.list <- list(c(1,2,3,4,25), 'apple', m1)
my.list
```

```
## [[1]]
## [1]  1  2  3  4 25
##
## [[2]]
## [1] "apple"
##
## [[3]]
##        amy     bob      charlie donna
## ages   "21"    "20"     "23"    "27"
## height "192"   "165"    "202"   "185"
## weight "65"    "102"    "85"    "78"
## word   "apple" "banana" "kiwi"  "orange"
```

Biztos, ami biztos nézzünk rá még egyszer erre a vektor dologra:

A következőben beteszünk egy ötelemű vektort, melynek értéke (1,2,3,4,25), második értékként beteszünk egy egyelemű vektort, melynek értéke 'apple', és beteszünk egy double típusú mátrixot.

Viszont most el is nevezzük őket! Az elsőnek neve lesz num.vector, másodiknak char.vector, harmadiknak matrix.

```
my.list <- list(num.vector=c(1,2,3,4,25), char.vector='apple', matrix=m1)
my.list
```

```
## $num.vector
## [1]  1  2  3  4 25
##
## $char.vector
## [1] "apple"
##
## $matrix
##        amy     bob      charlie donna
## ages   "21"    "20"     "23"    "27"
## height "192"   "165"    "202"   "185"
## weight "65"    "102"    "85"    "78"
## word   "apple" "banana" "kiwi"  "orange"
```

A lista elemeit név szerint a `$` operátorral érhetjük el:

```
my.list$num.vector
```

```
## [1]  1  2  3  4 25
```

A `q()` függvénnyel kiléphetünk az R konzolból. Y-mentsük a képet, N-ne, C-mégse lépjünk ki.

Szótár

kifejezés	expression
text1	text2
visszatérési érték	return value
változó	variable
értékadás	value assignment
értékadó operátor	value assignment operator
kiértékelés	evaluation
vektor	vector
összefűzés	concatenation
összefűz	concatenate
függvény	function
függvényhívás	function call
függvényhívó operátor	function call operator
argumentumok	argument
index	index
indexek	indices (ritkán indexes, google megérti azt is)
típus	type
atomi	atomic
primitív	primitive
összetett	composite
fölül ír	override
mátrix	matrix
mátrixok	matrices (ritkán matrixes, google megérti azt is)
implicit típuskonverzió	implicit type conversion
explicit típuskonverzió	explicit type conversion