Universitat Autònoma de Barcelona
Facultat de Ciències
Estadística
M. Saucedo

Seminari 1: introducció al R

A la pràctica d'avui farem un repàs de la sintaxi del R i intentarem veure algunes de les seves principals funcionalitats que aniran sortint al llarg de les properes sessions. Abans però, consulteu la introducció a R que teniu disponible al campus virtual per familiaritzar-vos amb la consola de R, l'editor R Studio i algunes de les comandes principals com és la utilització de la documentació. Per informació addicional podeu anar a la secció de referències de la introducció on trobareu documentació més completa.

Manipulacions i objectes bàsics

A la introducció hem vist com s'assignen variables. En aquest apartat, veurem alguns dels tipus d'objectes que podem assignar a les variables.

Vectors

Dins de tota la varietat d'objectes que proporciona R, un dels que més utilitzarem són els vectors. De vectors podem tenir-ne de 3 tipus: numèrics, de caràcters i lògics. Algunes maneres de declarar vectors són:

```
> x<-c(10.4,5.6,3.1,6.4,21.7)
> assign("x",c(10.4,5.6,3.1,6.4,21.7))
> x<-numeric(3)
> x
```

Per accedir a elements d'un vector seguim la mateixa sintaxi que en C: `x[i]`, per accedir a l'element que es troba a la posició i del vector x. En aquest cas, però cal tenir en compte que els índex dels vectors en R són ***1-based*** i que per tant l'element `x[0]` no existeix. Podem fer servir vectors ja creats per crear-ne de nous a partir de la **c**oncatenació:

```
> y <- c(x, 1, 1, x)
```

Per modificar elements particulars d'un vector només cal reassignar la variable,

```
> y[2] <-  5
```

Una altra manera de generar i declarar vectors és utilitzar les seqüències. Alguns exemples són:

```
> x<-1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x<-seq(1,10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x<-seq(1,10,by=.5);
> x
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0
```

```
[16]  8.5  9.0  9.5 10.0
> x<-seq(1,10, length=2)
> x
[1]  1 10
> x<-rep(x,3)
> x
[1]  1 10  1 10  1 10
```

R permet realitzar operacions entre vectors, les quals es duen a terme element per element. Observeu què passa si feu:

```
> 1/x
> sin(x)
> x+y
```

En canvi, també trobem funcions ja definides que reben com argument un vector multidimensional i retornen un escalar. Algunes d'elles són: `sum`, `mean`, `max`, `min`, `prod`, `length`... Mentre que d'altres reben i retornen vectors: `cumsum`, `sort`, `range`, `pmax`, `pmin`, ...

Els vectors numèrics també poden emmagatzemar valors complexos. Per declarar nombres complexos fem servir la variable i. Vegeu el següent exemple:

```
> sqrt(-4)
[1] NaN
Warning message:
In sqrt(-4) : NaNs produced
> sqrt(-4+0i)
[1] 0+2i
```

De la mateixa manera, podem tenir vectors de caràcters o strings. En R declarem els caràcters/strings utilitzant '". Per exemple:

```
> x <- c("Gos", "Gat", "Tortuga")
> z <- c(x, "Blau", "Vermell")
```

Proveu què passa si intenteu concatenar al vector `z` una seqüència de valors numèrics. De quin tipus són els elements del nou vector?

Per últim, trobem els vectors lògics que són vectors formats per elements `TRUE`, `FALSE` o `NA`. El següent exemple genera un vector d'elements lògics que conté FALSE a totes aquelles posicions on el valor emmagatzemat al vector x és més petit o igual que 10 i TRUE en cas contrari:

```
> x <- 1:20
> y <- x>10
> y
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Els operadors lògics en R que podem fer servir són:

- >, >=, <, <=, ==, !=
- & i |
- && i || són per seqüències i només comparen el primer element.

És molt habitual utilitzar la funció sum sobre vectors lògics per veure quants elements satisfan una condició:

```
> sum(x>10)
[1] 10
```

Una altra funció que s'utilitza molt és `is.na(x)`. Aquesta retorna TRUE o FALSE en funció si l'element del vector x és NA que s'utilitza per remarcar un problema a les dades. Per exemple:

```
> x <- c(seq(1,10), rep(NA,3), 1:2)
> is.na(x)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
[13]  TRUE FALSE FALSE
> sum(is.na(x))
[1] 3
```

Tots aquests operadors lògics i seqüències ens permeten accedir a diferents elements d'un vector per blocs per consultar el seu valor o simplement modificar el seu resultat. De la mateixa manera que accedíem a elements d'un vector utilitzant un índex de dimensió 1, podem utilitzar vectors numèrics o lògics per accedir a múltiples elements alhora. Vegeu que fan les següents instruccions:

```
> x[1:3]
[1] 1 2 3
> x[c(1,4)]
[1] 1 4
> x[-c(1,4)]
 [1]  2  3  5  6  7  8  9 10 NA NA NA  1  2
> x[!is.na(x)]
 [1]  1  2  3  4  5  6  7  8  9 10  1  2
> x[!is.na(x) & x > 5 ]
[1]  6  7  8  9 10
> x[is.na(x)] <- 0
> x[x<0] <- -x[x<0]
```

Factors

Un altre objecte particular de R són els factors. Aquests són especialment útils per representar dades categòriques com són el sexe, l'edat, la classe social, etc. Internament és un vector numèric amb valors compresos a $1, 2, \ldots, k$ on k és el nombre de nivells. Un factor amb k nivells s'emmagatzema internament com 2 elements: un vector format pels enters del 1 al k i un vector de caràcters que descriuen a que fan referència cadascun dels k nivells.

Per exemple, suposem que una enquesta la responen 100 dones i 200 homes. Una manera de representar això és utilitzant els factors:

```
> gender <- c(rep("dona", 100), rep("home",200))
> gender <- factor(gender)
> levels(gender)
[1] "dona" "home"
```

En aquest cas, el factor gender conté 100 1's que fan referència a les dones i 200 2's pels homes.

Veiem un altre exemple. Suposem que es realitza una enquesta a 5 persones que poden respondre del 1 a 5 en funció de la seva satisfacció amb el producte. Les respostes es guarden a

```
satisfaction <- c(1,3,4,2,2)
```

Voldrem crear un factor per tal de visualitzar i treballar millor amb aquesta informació. Podem fer el següent:

```
> fsatisfaction <- factor(satisfaction, levels=1:5)
> levels(fsatisfaction)
[1] "1" "2" "3" "4" "5"
> levels(fsatisfaction) <- c("molt dolent", "dolent", "normal", "bo", "molt bo")
> levels(fsatisfaction)
[1] "molt dolent" "dolent"      "normal"      "bo"          "molt bo"
> summary(satisfaction)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     2.0     2.0     2.4     3.0     4.0
> summary(fsatisfaction)
molt dolent      dolent      normal          bo     molt bo
          1           2           1           1           0
```

Matrius

Com és habitual, sovint haurem de treballar amb matrius. Les matrius en R es representem com vectors amb dimensions:

```
> x<-rnorm(20)
> x
 [1] -0.6343531 -1.5663532  0.4385617  0.6268317  0.5241078 -0.5515104
 [7]  0.0404109  0.6535566  2.1030837 -0.3835475  0.7261193  0.3762571
[13] -0.2416645  0.2045509  0.4798243 -0.9632663 -0.5614486 -0.3617397
[19]  0.7887358 -0.4585498
> dim(x)<-c(4,5)
> x
           [,1]       [,2]       [,3]       [,4]       [,5]
[1,] -0.6343531  0.5241078  2.1030837 -0.2416645 -0.5614486
[2,] -1.5663532 -0.5515104 -0.3835475  0.2045509 -0.3617397
[3,]  0.4385617  0.0404109  0.7261193  0.4798243  0.7887358
[4,]  0.6268317  0.6535566  0.3762571 -0.9632663 -0.4585498
```

Noteu que per defecte R ompla la matriu per columnes. Això es pot canviar utilitzant la funció `matrix()`.

```
> x<-rnorm(20)
> x
 [1]  0.04173864  0.10018925  0.29070242 -1.31307907 -1.89205762 -1.15747131
 [7]  0.59374516  0.93460942 -1.75483040 -0.45234251 -0.24609229 -0.83206577
[13]  1.18091794 -0.92651117  1.01319847 -0.29973138 -0.01365092  1.13642107
[19]  0.26082700 -0.17589814
> matrix(x, nrow=4, ncol=5, byrow=T)
            [,1]        [,2]      [,3]       [,4]       [,5]
[1,]  0.04173864  0.10018925 0.2907024 -1.3130791 -1.8920576
[2,] -1.15747131  0.59374516 0.9346094 -1.7548304 -0.4523425
[3,] -0.24609229 -0.83206577 1.1809179 -0.9265112  1.0131985
[4,] -0.29973138 -0.01365092 1.1364211  0.2608270 -0.1758981
```

Proveu algunes funcions que apliquen sobre matrius com són: `nrow()`, `ncol()`, `t()`, `rownames()`, `colnames()`, etc.

Una altra manera de guardar matrius és a partir de la concatenació d'altres ja definides. Aquesta combinació és pot realitzar per files o columnes i les funcions que s'encarreguen d'aixo són `rbind()` i `cbind()`.

```
> x <- matrix(1:10, nrow=2, ncol=5)
> y <- matrix(11:18, nrow=2, ncol=4)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> y
     [,1] [,2] [,3] [,4]
[1,]   11   13   15   17
[2,]   12   14   16   18
> cbind(x,y)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    3    5    7    9   11   13   15   17
[2,]    2    4    6    8   10   12   14   16   18
> rbind(x,y)
Error in rbind(x, y) :
  number of columns of matrices must match (see arg 2)
```

De la mateix manera que amb els vectors, les operacions sobre matrius actuen element per element per defecte. Per realitzar per exemple el producte matricial de dues matrius s'ha d'especificar utilitzant

```
A %*% B
```

Per últim, per accedir als elements d'una matriu fem servir un índex doble. En aquest cas, anàlogament amb els vectors, també podem utilitzar seqüències per accedir a múltiples elements. Observeu els següents exemples.

```
> x <- matrix(rnorm(16), ncol=4, nrow=4)
> x
           [,1]        [,2]       [,3]       [,4]
[1,] -1.6427314  0.32043856 -0.3917217 -1.6137698
[2,]  0.5456363  0.78959052  0.5502112  1.0856365
[3,] -0.1966351 -0.41549753  0.8227324 -0.4110604
[4,] -0.7832441 -0.06042838  1.0117765 -1.5359515
> x[3,2]
[1] -0.4154975
> x[3,1:2]
[1] -0.1966351 -0.4154975
> x[,c(1,3)]
           [,1]       [,2]
[1,] -1.6427314 -0.3917217
[2,]  0.5456363  0.5502112
[3,] -0.1966351  0.8227324
[4,] -0.7832441  1.0117765
> x[-1,-3]
           [,1]        [,2]       [,3]
[1,]  0.5456363  0.78959052  1.0856365
```

```
[2,] -0.1966351 -0.41549753 -0.4110604
[3,] -0.7832441 -0.06042838 -1.5359515
```

Llistes

Com hem vist anteriorment, els vectors són seqüències d'elements d'un mateix tipus. Aquesta característica limita en moltes ocasions el seu ús, on sovint necessitarem utilitzar seqüències no homogènies. En aquest cas, podem fer ús d'un altre tipus d'objecte com són les llistes. Una llista és una col·lecció ordenada d'elements que poden ser objectes arbitraris. Les llistes es creen fent servir la funció `llist()` i a cada element de la llista se li pot assignar un nom. Aquests elements són accessibles fent servir un o dos parells de brackets. Alguns exemples sobre accedir elements d'una llista:

```
> L <- list(nom="Alex", edat=30, n.fills=2, nom.fills=c("Ivan","Ruben"))
> L[["nom"]]
[1] "Alex"
> L[[1]]
[1] "Alex"
> L$nom
[1] "Alex"
> names(L)
[1] "nom"       "edat"      "n.fills"   "nom.fills"
```

Dues llistes es poden concatenar de la mateixa manera que es fa amb els vectors.

Data frames

L'últim objecte que veurem seran els data frames. Aquests els podem pensar com conjunts o matrius de dades generalitzades en dues dimensions. Aquests són una llista de vectors o factors amb el mateix nombre d'entrades i, per tant, només té un únic conjunt de noms de files: un per a cada registre. Veiem algunes maneres de crear un data frame:

```
> d <-
data.frame(mpes=c(71.5,72.1,73.7,74.3,75.2,74.7),sexe=c("M","M","F","F","M","M"))
> d1 <- as.data.frame(llista)
> d2 <- data.frame(matrix(vector(), 0, 4))
```

La manera d'accedir als elements d'un data frame és semblant a la que s'utilitza amb les matrius i vectors, essent possible també utilitzar operadors lògics i l'operador $ per accedir pel nom de registre. Observeu les següents instruccions:

```
> data(faithful)
> names(faithful)
[1] "eruptions" "waiting"
> summary(faithful)
> faithful[1,2]
[1] 79
> faithful[2,]
> faithful[faithful$eruptions<4,]
```

Exercici 1 Vectors:

1. Declareu un vector que tingui les entrades següents

```
3 4 1 5 8 7 1 5 3 5 0 1 9 0 5 8 9
```

i comproveu quins dels seus elements són iguals a 5. Modifiqueu el vector que heu declarat per tal que els 5's siguin 0's.

2. Creeu un nou vector que contingui tots elements del vector anterior que siguin més grans que 3.
3. Creeu una seqüència de nombres des del 2 al 20 amb pas 0.4.
4. Concateneu els vectors de l'apartat (1) i (2) per crear un nou vector i guardeu-lo.
5. Elimineu les variables que hagueu creat en el vostre workspace.
6. En un estudi, sis pacients van quantificar el seu dolor del 0 al 3, essent 0 *cap color*, 1 *baix dolor*, 2 *dolor mitjà* i 3 *dolor agut*. Els resultats es mostren a la taula següent

Pacient	1	2	3	4	5	6
Dolor	0	3	1	1	0	2

declareu un factor `f` per representar-los.

Exercici 2 Matrius:

1. Declareu dues matrius A i B de dimensions 4×3 i 1×3 utilitzant la funció `rnorm()` i ompliu-les per files.
2. Definiu una nova matriu C a partir de A i B combinant-les ja sigui per files o per columnes.
3. Determineu la dimensió de la nova matriu fent servir `dim()`.
4. Calculeu

$$\begin{pmatrix} 2 & -1 & 4 \\ 0 & 3 & -2 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 1 & 7 \end{pmatrix}$$

i imprimiu la segona columna de la matriu resultant.

Exercici 3 Llistes:

1. Creeu quatre vectors amb les següents entrades:

Any	1980	1988	1996	1998	2000	2002
Pes	71.5	72.1	73.7	74.3	75.2	74.7
Sexe	M	M	F	F	M	M
Alçada	179.3	179.9	180.5	180.1	180.3	180.4

2. Declareu una llista que contingui els següents vectors i doneu a cada component d'aquesta un nom concret.
3. Feu servir 3 maneres diferents per accedir al quart element de la llista.

Exercici 4 Data frames:

1. Creeu un data frame amb la següent informació

Nom	Cognom	Edat	Sexe	Punts
Alice	Ryan	37	F	278
Paul	Collins	34	M	242
Jerry	Burke	26	M	312
Thomas	Dolan	72	M	740
Marguerite	Black	18	F	177
Linda	McGrath	24	F	195

2. Guardeu els punts de cada persona en un vector i calculeu la mitjana de punts.
3. Guardeu en un altre vector els punts de totes les persones de menys de 35 anys.
4. Determineu l'edat màxima de tots els registres.
5. Extraieu tots aquells registres tals que tinguin més de 200 punts i menys de 30 anys.

Llegir i escriure fitxers

A continuació, revisarem com llegir i escriure dades des de fitxers externs, ja siguin full de càlcul, fitxers de text o bé directament des de servidors. En aquests casos, disposem de diverses funcions que permeten fer-ho de manera relativament senzilla i directa. Consulteu per exemple les funcions `scan()`, `read.table()` i `read.csv()` que s'encarreguen de llegir dades. Per exemple:

```
> adr <- 'https://raw.githubusercontent.com/Miquelsc/est/main/nadons.txt'
> nadons <- read.table(file=adr,  header=T, sep=',')
```

Podeu comprovar el tipus d'objecte que és nadons utilitzant les funcions `is.matrix()`, `is.data.frame()`, etc. Quina diferència hi ha entre la funció que acabem d'utilitzar i `read.csv()`?

De la mateixa manera, es pot escriure a fitxers utilitzant les funcions `write.csv()` o `write.table()` (consulteu l'ajuda per veure en què es diferencien). Per exemple, podem guardar la taula que hem llegit des d'un servidor en un fitxer a la nostra màquina local executant:

```
> write.table(nadons, file='nadons.txt', row.names=F, sep=' ')
```

A la darrera instrucció no hem especificat el path del fitxer que volem desar. Per defecte, aquest es guarda el *current working directory* on s'inicia la sessió de R:

```
> getwd()
[1] "/home/convidat"
```

Per canviar-lo, podeu utilitzar,

```
> setwd("/home/convidat/Documents")
```

Ara, la instrucció anterior desarà el fitxer `nadons.txt` al directori Documents. També podeu especificar el *full path* o *relative path* quan crideu la funció `write.table()`[1]:

```
> write.table(nadons, file='/home/convidat/Documents/nadons.txt', row.names=F,
sep=' ')
> write.table(nadons, file='../nadons.txt', row.names=F, sep=' ')
```

Exercici 5 Llegint dades des de fitxer:

[1] Alerta. Aquestes instruccions estan escrites suposant que treballeu amb un sistema basat en Linux. Per especificar els path en Windows heu de canviar els / per \: `'C:\Users\convidat\Desktop'`

1. Descarregueu o llegiu directament el fitxer que trobareu a `https://raw.githubusercontent.com/Miquelsc/est/main/malaria.txt` i guardeu-lo al vostre workspace.
2. Consulteu els noms de les columnes.
3. Genereu dues taules diferents: una per les dades dels homes i una altra per les de les dones.
4. Calculeu la mitjana, la desviació estàndard, el màxim i el mínim de l'edat pels dos casos.

Simulacions i distribucions de probabilitat

R és un llenguatge de programació especialment adient per realitzar càlculs estadístics i, com a tal, disposa d'una gran varietat de funcions per simular distribucions de probabilitat i realitzar test de distribucions. Algunes són

- `norm`: normal.
- `t`, `chisq`, `f`: tests normals.
- `unif`: uniforme.
- `gamma`, `cauchy`, etc.
- `binom`, `pois`, `negbin`, etc.

A aquestes instruccions se li afegeixen els prefixos `p`, `q`, `d` o `r` en funció del que es vulgui calcular: funció de densitat, quantil, probabilitat, etc. Podeu consultar l'ajuda de `pnorm` per veure què fa cada instrucció en funció del prefix.

Una altra instrucció molt útil que trobem a la llibreria es diu `sample()`. Aquesta tria nombres aleatoris d'una mostra seleccionada i ho pot fer amb o sense reemplaçament. Així, cada vegada que executem la instrucció, R pot triar diferents nombres i, per tant, dues execucions d'un mateix script poden tenir resultats diferents. Això es pot solucionar fent servir la comanda `set.seed()`. Vegeu el següent exemple per entendre com funciona

```
> sample(1:20, 5)
[1]  7  1 17 19  3
> sample(1:20, 5)
[1] 18 17 16 12 15
> set.seed(9)
> sample(1:20, 5)
[1]  6 19  3 12 16
> set.seed(9)
> sample(1:20, 5)
[1]  6 19  3 12 16
```

Exercici 6 Simulacions i distribucions de probabilitat.

1. Genereu una mostra aleatòria sense reemplaçament i de mida 100 utilitzant la funció `sample` a partir de l'interval $[0, 2]$ que conté 201 elements equiespaiats.
2. Feu servir la funció `dt` per avaluar la funció de densitat d'una distribució t amb 13 graus de llibertat en una successió de 21 valors equiespaiats a l'interval $[-1, 1]$.
3. Troba el valor de x tal que $P(X \leq x) = 0{,}01$ per a una t distribució amb 9 graus de llibertat (`qt()`).
4. Els resultats d'un test de coeficient intel·lectual (IQ) segueixen una distribució normal de mitjana 100 i distribució estàndard 15. Quina és la probabilitat de tenir un IQ de sota de 142?.

Dibuixant gràfiques

A continuació farem un breu resum d'algunes de les funcions que disposa R per representar gràficament resultats (plots). R té algunes funcions que generen els plots principals com són `plot()`, `hist()`, `boxplot()`, etc. A aquests plots se li poden afegir gràfics addicionals mitjançant les funcions `points()`, `lines()`, `abline()`, `legend()`, etc. Per veure més informació podeu utilitzar

```
> demo(graphics)
```

En general farem servir la funció `plot()` de la qual podeu consultar alguns dels arguments que pot rebre a l'ajuda. Amb aquesta instrucció podem dibuixar funcions, per exemple:

```
> par(mfrow=c(1,3))
> x<-seq(0,1,length=20)
> plot(sin(2*pi*x))
> plot(sin(2*pi*x),type="l")
> plot(sin(2*pi*x),type="b")
```

També es poden generar gràfics a partir de data frames o matrius. Per exemple:

```
> par(mfrow=c(1,1))
> library(MASS)
> ?Cars93
> plot(Cars93$Weight,Cars93$EngineSize,ylab="EngineSize"
   +,xlab="Weight",main="Myplot")
> lines(x=c(min(Cars93$Weight),max(Cars93$Weight)),
   + y=c(min(Cars93$EngineSize),
   + max(Cars93$EngineSize)),lwd=4,lty=3,col="green")
> abline(h=3,lty=2)
> abline(v=1999,lty=4)
> fm1<-lm(EngineSize~Weight,Cars93,
   +subset=Origin=="USA")
> abline(coef(fm1),lty=4,col="red")
```

En el cas que es desitgi extreure histogrames, es pot fer servir la funció `hist()`,

```
> par(mfrow=c(1,2))
> hist(USA.weight,breaks=10,xlim=c(1500,4500),col="grey")
> hist(nonUSA.weight,breaks=10,xlim=c(1500,4500))
```

i el mateix per boxplots,

```
> boxplot(Cars93$Weight)
```

Exercici 7 Dibuixant gràfiques.

1. Declareu un vector x que contingui una seqüència de 20 valors de l'1 al 20.
2. Declareu un altre vector `w <- 1+sqrt(x)/2`.
3. Creeu un data frame que tingui dues columnes, una `x=x` i en una altra de nom `y` els resultats de `x+rnorm(x)*w`.
4. Dibuixeu un histograma i un boxplot de la variable `y` en la mateixa finestra.
5. Dibuixeu els valors de la columna y contra els de la x de la manera apropiada.
6. Feu servir la comanda `lm()` i `abline()` per ajustar una regressió lineal i afegir-la al gràfic anterior.

Estructures de control i funcions

Per últim, veure'm com es poden declarar noves funcions. Al llarg de la sessió hem vist diferents funcions que estan ja declarades a les llibreries que disposa R (`sum()`, `min()`, `mean()`, ...). Aquests són mètodes que reben una llista d'arguments i retornen diferents objectes, ja siguin valors escalars, vectors, data frames, etc. La sintaxis per declarar noves funcions a R és la següent:

```
function_name <- function (arg1, arg2, ...) {
   commands
   output
}
```

Un exemple molt senzill:

```
> pow2 <- function(x) {
+ x^2
+ }
> pow2(4)
[1] 16
> pow2(1:4)
[1]  1  4  9 16
```

De la mateixa manera que passava amb les variables, les funcions declarades es poden veure utilitzant la comanda `ls()`. Una altra característica de les funcions de R és que permeten introduir els arguments pel nom de la variable que reben dins de la funció i sense respectar l'ordre amb el que s'han definit. Consulteu el següent exemple:

```
> opr <- function (x, y, z) {
+ x+2*y+3*z
+ }
> opr(1,2,3)
[1] 14
> opr(z=3, y=2, x=1)
[1] 14
```

També es permet la definició de valors per defecte, els quals s'utilitzen quan no es passa cap argument a la funció:

```
> mypow <- function (x, pow=2) {
+ x^pow
+ }
> mypow(2,3)
[1] 8
> mypow(2)
[1] 4
```

Fins ara hem vist casos en els quals les funcions retornaven valors escalars. Tanmateix, es possible definir funcions que retornin més d'un objecte i de tipus diferent. Per fer això, es fan servir les llistes:

```
> exemple <- function (x) {
+ the.mean <- mean(x)
+ the.sd <- sd(x)
+ the.min <- min(x)
```

```
+ the.max <- max(x)
+ return(list(average=the.mean, stand.dev=the.sd, minimum=the.min, maximum=the.max))
+ }
> res <- exemple(rnorm(20))
> res$average
[1] 0.1921187
```

Exercici 8 Escriviu les funcions següents:

1. Escriviu una funció que rebi un nombre i retorni el seu quadrat, el seu cub i la seva arrel quadrada. La funció ha de poder rebre valors negatius.
2. Escriviu una funció que rebi un vector numèric i imprimeixi per pantalla el màxim, el mínim i la mitjana i, finalment, dibuixi el seu histograma.
3. Escriviu una funció que calculi la suma de les longituds de tres vectors.

A l'hora de definir funcions són molt importants les estructures de control de flux: les execucions condicionals, els loops, etc. A continuació introduirem la seva sintaxi i veurem alguns exemples.

En primer lloc, veurem les sentències condicionals: aquestes poden ser del tipus if-else o bé ifelse.

```
if (condition) {
   expr_1
} else {
   expr_2
}
```

```
if (condition 1 ) {
   expr_1
} else if (condition 2) {
   expr_2
}
...
else {
   expr_n
}
```

En aquest cas, cada parèntesi conté una instrucció lògica que ha de retornar TRUE o FALSE.

Una altra estructura de control bàsica són les sentències iteratives, en els quals es repeteix una mateixa instrucció en funció de certes condicions. Un exemple d'aquest tipus de sentència són els loops for:

```
for (variable in sequence) {
   command
}
```

Un altre exemple són els loops while:

```
while (condition) {
   command
}
```

Per últim, anem a veure que R inclou tres funcions anomenades `apply()`, `lapply()` i `sapply()` que són molt més eficients que els loops quan es volen realitzar operacions repetides sobre vectors, llistes, matrius o data frames. En el cas de la funció `apply()` es crida

```
apply(vector, way, function)
```

on el primer argument és la matriu o data frame sobre el qual es vol calcular, el segon argument indica com es vol executar l'operació si sobre files (1) o columnes (2) i, per últim, el tercer argument indica la funció que es vol aplicar. Vegeu el següent exemple:

```
> rmatrix <- matrix(1:20, nrow=2)
> dim(rmatrix)
[1]  2 10
> apply(rmatrix, 1, sum)
[1] 100 110
> apply(rmatrix, 2, sum)
 [1]  3  7 11 15 19 23 27 31 35 39
```

La mateixa idea s'aplica amb les funcions `lapply()` i `sapply()` quan es volen realitzar operacions sobre llistes i vectors respectivament. Observeu els següents exemples i fixeu-vos què retorna cada funció en cadascun dels casos:

```
> lapply(c(1,2,3), function(x){return (x*2)})
> sapply(1:20, function(x) { x-1 })
```

Exercici 9 Estructures de control.

1. Escriviu utilitzant un for un programa que calculi i emmagatzemi en un vector y totes les potències de 2 d'un vector que contingui tots els naturals del 1 al 50000. Penseu dues maneres de més de realitzar el mateix càlcul i compareu els temps d'execució fent servir la funció `Sys.time()`.
2. Escriviu un programa que calculi la suma dels naturals fins que aquesta sigui més gran que 100. Modifiqueu el darrer programa per tal que es pugui construir un vector amb les sumes parcials (**Hint:** consulteu la instrucció `repeat`).

Referencias

[1] Gonzalez, Juan R, (2019) *Based on material by Dr.Norma Bargary, Univeristy of Limerick*