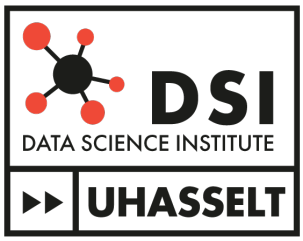

Introductie programmeren in R

Dag van de wetenschap & wiskunde - KULAK Kortrijk

Wouter Smeets - wouter.smeets@uhasselt.be
Miguel Beynaerts - miguel.beynaerts@uhasselt.be
Steven Abrams - steven.abrams@uhasselt.be

16 November 2024

Contents

Chapter 1

Hallo

Chapter 2

Wat is R?

In deze cursus zal het gratis softwarepakket *R* gebruikt worden. *R* is geschikt voor het manipuleren, statistisch analyseren en visualiseren van data. Daarnaast zullen we gebruikmaken van RStudio, een geïntegreerde ontwikkelomgeving voor R.

2.1 R software downloaden

De nieuwste versie van *R* (versie 4.4.2) kan gedownload worden via volgende link: https://cran.r-project.org/bin/windows/base/.

2.2 RStudio software downloaden

Hierna kan **RStudio** geïnstalleerd worden via volgende link: https://posit.co/download/rstudio-desktop/.

RStudio wordt gezien als een gebruiksvriendelijkere versie van de standaard R software met een modernere interface. De programmeertaal is echter volledig identiek in beide programma's.

2.3 Waarom R?

R is een programmeertaal die de laatste jaren heel veel populariteit heeft gewonnen. Dit komt enerzijds door de toegankelijkheid en gebruiksvriendelijkheid naar beginners toe van de programmeertaal zelf, maar anderzijds ook door de vele mogelijkheden die de taal biedt. Zo is het geschikt voor het bewerken, het

visualiseren en het analyseren van de data. Op onderstaande figuur is zichtbaar hoe populair de verschillende software talen waren in 2018 voor een bepaalde groep onderzoekers.

2.4 Het R dashboard

Zodra RStudio geopend wordt, verschijnt een 3-delig venster. In de *Environment* (venster rechtsboven) kan je vinden welke variabelen en datasets in de huidige R sessie geladen zijn. Dit is momenteel leeg, maar zal in de loop van deze cursus aangevuld worden. In het paneel rechtsonder staat automatisch het venster *Plots* open waarin figuren verschijnen wanneer dit gevraagd wordt. Dit zijn de twee hulpvensters. In het venster aan de linkerkant staat de **Console** open waarin alle commando's en output verschijnen die recent zijn uitgevoerd.

In deze Console kan je zelf commando's typen achter de '>'. Bijvoorbeeld, als je de som 3 + 4 willen berekenen, zet je deze in de Console en druk je op ENTER.

```
3 + 4
```

```
## [1] 7
```

De uitkomst van deze som komt ook in de Console terecht, onder de ingevoegde bewerking. Ook eventuele foutmeldingen die je later kan tegenkomen, zullen in deze Console verschijnen.

2.5 Het R script

Om te voorkomen dat je commando's steeds opnieuw moet typen in de console, of wanneer je de commando's wilt opslaan, kan een **script** aangemaakt worden.

Om dit te doen, ga je naar File > New File > R Script en dan verschijnt er een vierde venster linksboven in de sessie, boven de Console. Dit venster heeft voorlopig als naam 'Untitled 1'.

In dit script kan je nu verschillende lijnen code typen die uitgevoerd zullen worden. Bijvoorbeeld, typ onder elkaar de commando's 1 + 1 en 3 + 4. Klik dan op de knop 'Run' (dit is het icoontje rechtsboven in je script met het groene pijltje). Een sneltoets om code te runnen is 'CTRL + ENTER' (in geval van een Windows toestel) of 'Cmd + ENTER' (in geval van Mac toestellen). Deze shortcut (en een overzicht van andere shortcuts) kan je terugvinden in (bijv.) Code > Run Selected Line(s).

```
1 + 1
```

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

```
3 + 4
```

```
## [1] 7
```

Om een bestaand R script te openen, klik je op File > Open File en vervolgens navigeer je naar de map waarin het bestand staat en kies je het juise bestand (met extensie .R) dat je wilt openen.

Om het script op te slaan, ga je naar File > Save As en vul je een bestandsnaam naar keuze in. Let wel op dat je als extensie .R kiest.

Het is ook mogelijk om **commentaar** te typen bij stukjes code. Dit doe je door eerst een # en daarachter code te typen. Deze lijn wordt dan niet uitgevoerd bij het runnen van de code. Dit kan handig zijn om extra uitleg te schrijven achter bepaalde functies (bijvoorbeeld met betrekking tot de betekenis van diens argumenten) of bij een deel van de code. In het volgende stukje code zal het eerste deel wel uitgevoerd worden, maar het tweede deel niet.

```
3 + 4
```

```
## [1] 7
```

```
# 5 + 6, dit stukje wordt niet uitgevoerd door de '#'
```

Chapter 3

Commando's in R

R kan alles wat een zakrekenmachine ook kan. Enkele voorbeelden worden hieronder gegeven:

```
2+3 # Optellen door '+'
```

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

```
2*3 # Vermenigvuldigen door '*'
```

```
## [1] 6
```

```
2^3 # Machten berekenen door '^'
```

```
## [1] 8
```

```
9-3 # Aftrekken door '-'
```

```
## [1] 6
```

```
9/3 # Delen door '/'
```

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

De vierkantswortel van x kan via het *sqrt(x)* commando verkregen worden:

```
sqrt(16)
```

```
## [1] 4
```

Ook de logaritmische en exponentiële functies kunnen gebruikt worden. De exponentiële functie e^x werkt via het commando *exp(x)* en de logaritmische functie $log_a(x)$ via *log(x, a)*.

De exponentiële functie in 4 geëvalueerd en de logaritme van 3 met grondtal 7 worden dus bekomen door:

```
exp(4)
```

```
## [1] 54.59815
```

```
log(3, 7)
```

```
## [1] 0.564575
```

Let op: de functie $\log(x)$ evalueert standaard (i.e. als je geen a specifieert) de natuurlijke logaritme met grondtal e in x, m.a.w. is dit eigenlijk de functie $\ln(x)$. De volgende twee commando's hebben daarom dezelfde oplossing.

```
log(4)
```

```
## [1] 1.386294
```

```
log(4, exp(1))
```

```
## [1] 1.386294
```

Verder kan je de uitkomst van bewerkingen ook opslaan in nieuwe variabelen. Dit doe je door '=' of '<-' te gebruiken. Als je bijvoorbeeld de uitkomst van een optelling wilt toewijzen aan een nieuwe variabele genaamd *nieuw*, kan dat als volgt:

```
nieuw <- 3 + 4
```

Merk op dat er geen output verschijnt. In het *Environment* venster rechtsboven verschijnt de nieuwe variabele wel met zijn waarde. Om via de *Console* te kijken wat de waarde van deze nieuwe variabele is, kan je hem gewoon in de Console ingeven:

```
nieuw
```

```
## [1] 7
```

Nu zie je dat deze variabele de waarde 7 heeft. Stel dat je de variabele een andere waarde wil geven, dan kan je deze gewoon overschrijven met een nieuwe bewerking:

```
nieuw <- 5*2
nieuw
```

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

Nu heeft de variabele een nieuwe waarde gelijk aan 10 en is hij de oude vergeten. Je kan een variabele elke naam geven die je wilt.

LET OP: R is hoofdlettergevoelig. Houd hiermee rekening in het kiezen van namen en het uitvoeren van commando's; het is aangewezen om via een consistente manier de naamgeving te kiezen om problemen te vermijden. Let verder ook op bij het kiezen van namen die gelijk zijn aan bestaande R functies. Zo wordt de letter t ook gebruikt voor de zogeheten student t-verdeling, is *mean* de naam van een functie in R die het gemiddelde berekent en zijn woorden zoals *TRUE* of *FALSE* de ingebouwde namen voor logische operatoren in R.

Verder is het ook mogelijk om je aangemaakte variabelen in nieuwe bewerkingen te gebruiken:

```
sqrt(nieuw)
```

```
## [1] 3.162278
```

```
nieuw*2
```

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

```
(nieuw+nieuw)/4
```

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

Als je meer info wilt krijgen over een bepaald commando, dan kan je er een '?' voor zetten in de Console. Vervolgens wordt er rechtsonder in het venster 'Help' een hulpvenster geopend met meer info. Bijvoorbeeld, voor extra informatie over het gebruik van het *log*-commando, kan je het volgende in de Console typen:

```
?log
```

Om te laten zien waarom het maken van een script handig kan zijn, kan je R eens even afsluiten (vergeet niet op opslaan te klikken als je dit nog niet hebt gedaan!). Als je R nu opnieuw opent, zie je dat het script er nog steeds is, maar dat de Console terug leeg is. Als je je werk dus wilt bewaren, werk je best in een script i.p.v. de Console.

Chapter 4

Data types en structuren in R

Er zijn verschillende **data types** die gebruikt worden in R:

- Numeric
- Character
- Factor
- Logical

Elementen van bovenstaande datatypes kunnen gecombineerd worden in **data structuren**:

- Vector
- Data Frame
- Matrix
- List

4.1 Numeric data type

Numerieke elementen zijn elementen met een getalwaarde. Met numerieke elementen kunnen bewerkingen uitgevoerd worden die opnieuw resulteren in een nieuw numeriek element.

Om te controleren welk data type een bepaalde variabele heeft, kan de *str* functie gebruikt worden.

```
a <- 5
b <- 6
str(a)
```

```
##  num 5
```

```
str(b)
```

```
##  num 6
```

```
a + b
```

```
## [1] 11
```

```
str(a + b)
```

```
##  num 11
```

Binnen de klasse van numerieke elementen is er echter nog een opsplitsing van twee data types. Zo maakt R een onderscheid tussen gehele getallen en in het algemeen reële getallen die dus ook decimalen kunnen bevatten. R beschouwt getallen standaard als reëele getallen. Het element heeft dan data type *dbl* of *double*, wat staat voor *double precision*, duidend op het feit dat het ook decimale getallen kan aannemen als waarde. Gehele getallen, hebben het data type *int* (van het Engels: integer). Zij worden gespecifieerd door middel van de letter *L* achter het getal te plaatsen. Om het exacte data type na te gaan, kan de functie *typeof()* gebruikt worden. Een voorbeeld van de twee verschillende data types:

```
c <- 1000L #Dit beschouwt R als een geheel getal.
typeof(c)
```

```
## [1] "integer"
```

```
q <- 1000 #Dit beschouwt R als een reëel getal.
typeof(q)
```

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

In het algemeen maakt dit verschil niet heel veel uit als je bewerkingen wilt uitvoeren. Om elementen van het type *dbl* naar het type *int* om te zetten, kan je het commando *as.integer* gebruiken. Om elementen van het type *int* naar het type *dbl* om te zetten, kan je het commando *as.numeric* gebruiken.

4.2 Character data type

Characters zijn elementen in de vorm van letters of stukken tekst. Deze elementen worden altijd omgeven met aanhalingstekens.

```
d = "Goedemiddag"
f = "allemaal"

str(d)
```

```
##  chr "Goedemiddag"
```

Merk op dat met characters geen bewerkingen uitgevoerd kunnen worden. Dit zou resulteren in een Error:

```
d + f
```

```
## Error in d + f: non-numeric argument to binary operator
```

Het is wel mogelijk om verschillende characters te printen. Dit gebeurt via de *paste* functie waarin je de elementen specifieert die je naast elkaar wilt printen en waarin je in het argument *sep=* specifieert welke scheiding je wilt tussen de verschillende elementen.

```
paste(d, f, sep=',')
```

```
## [1] "Goedemiddag,allemaal"
```

4.3 Vector data structuur

Elementen van de 2 vorige data types, numeriek en character, kunnen ook in een **vector** gestoken worden. Een vector is een 1-dimensionaal data structuur object dat enkel elementen bevat van hetzelfde data type. Je kan een vector aanmaken met behulp van 'c()' waarbij je tussen de haakjes alle elementen in je vector opsomt, gescheiden door een komma.

```
g <- c(1, 4, 7, 10)
h <- c('Aap', 'Mug', 'Olifant')
```

Een vector die enkel numerieke elementen bevat, kan ook in bewerkingen gebruikt worden. De bewerkingen worden dan elementsgewijs uitgevoerd.

```
g*4
```

```
## [1]  4 16 28 40
```

```
g^2
```

```
## [1]   1  16  49 100
```

```
g-4
```

```
## [1] -3  0  3  6
```

Het is ook mogelijk om alle gehele getallen tussen 2 grenzen meteen te specifiëren. Dit kan met behulp van het dubbel punt-commando. Zo zal het commando *1:5* alle getallen van 1 tot en met 5 geven.

```
m <- 1:5
m
```

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

```
n <- 67:98
n
```

```
##  [1] 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
## [26] 92 93 94 95 96 97 98
```

Merk op dat je bij uitzondering voor dit specifieke commando niet het *c()* commando nodig hebt.

Daarnaast kan je ook een sequentie van getallen als vector creëren met daarin een minimum, maximum en stapgrootte (bijvoorbeeld: alle getallen tussen 0 en 20 met stapgrootte 2 of alle getallen tussen 4 en 50 met stapgrootte 6):

```
s <- seq(from = 0, to = 20, by = 2)
s
```

```
##  [1]  0  2  4  6  8 10 12 14 16 18 20
```

```
seq(4, 50, 6)
```

```
## [1]  4 10 16 22 28 34 40 46
```

Om een aftellende vector te verkrijgen, start je met het grootste getal te specifiëren. Vervolgens specifieer je bij de stapgrootte nu een minteken voor het getal om aan te duiden dat je stappen wilt zetten van groot naar klein. Een voorbeeld:

```
seq(40, 7, -4)
```

```
## [1] 40 36 32 28 24 20 16 12  8
```

4.4 Matrix data structuur

Een **matrix** data structuur is een 2-dimensionaal object dat, net zoals vectoren, enkel elementen van eenzelfde data type bevat. Een matrix wordt gemaakt via de *matrix()* functie waarin gespecifieerd wordt welke waarden in de matrix moeten komen en hoeveel rijen en/of kolommen de matrix heeft. In het volgende voorbeeld wordt een matrix met elementen 1 tot en met 12 gemaakt waarbij de matrix 4 rijen heeft (en dus 3 kolommen).

```
matrix(1:12, nrow = 4)
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Merk op dat de matrix de elementen kolomgewijs toewijst. Stel dat we dit rijgewijs willen doen (dus zodat 1 2 3 in de eerste rij staan in plaats van de eerste kolom), kan het extra argument *byrow = TRUE* gebruikt worden.

```
matrix(1:12, nrow = 4, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

4.5 List data structuur

Een **list** data structuur is een combinatie van andere soorten structuren. Zo kan het elementen van verschillende soorten data types bevatten, maar ook andere data structuren zoals een vector, matrix of verschillende andere lists. In het volgende voorbeeld beschouwen we een lijst met een numerieke waarde, vector en matrix als elementen.

```
mijnlijst <- list(a = 1:5, b = matrix(1:9, ncol=3), c = 10)
mijnlijst
```

```
## $a
## [1] 1 2 3 4 5
##
## $b
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $c
## [1] 10
```

Individuele elementen van een lijst kunnen opgeroepen worden zoals een variabele in een dataframe met het *$* teken, ofwel via [[$k$]] waarbij het k-de element van de list wordt opgeroepen.

```
mijnlijst$b
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
mijnlijst[[2]]
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Lijsten zijn uiterst handig bij het gebruik van functies waar meerdere elementen van verschillende types als uitvoer moeten gegeven worden. Het gebruik van lijsten is bovendien computationieel efficiënt.

4.6 Dataframe data structuur

Een vierde en laatste belangrijke data structuur is de **dataframe**. Dit is een 2-dimensionele dataset die in de kolommen de variabelen weergeeft en in de rijen de observaties (individuen, dieren, ...) zelf. In het volgende voorbeeld wordt een dataframe gemaakt met daarin de namen van 3 personen en hun leeftijd. Dit dataframe bevat dus een character en numeriek type variabele.

```
dataframe1 <- data.frame (
  Naam = c("Jan", "Marie", "Koen"),
  Leeftijd = c(22, 15, 19)
)

dataframe1
```

```
##    Naam Leeftijd
## 1   Jan       22
## 2 Marie       15
## 3  Koen       19
```

Op dataframes (en hoe je die zelf kan maken of inlezen) wordt later nog teruggekomen.

4.7 Factor data type

Een **factor** element is een categorische variabele. Oorspronkelijk kan dit element een numeriek of character type geweest zijn, dat omgezet is naar een factor variabele.

Er wordt nu eerst een vector gecreëerd met numerieke elementen en een vector met character elementen.

```
a = c(1, 4, 8)
b = c("Groen", "Blauw", "Rood")
```

Vervolgens kunnen deze vectoren omgezet worden naar variabelen van het type factor:

```
factor1 <- factor(a)
factor2 <- factor(b)

str(a)
```

```
##  num [1:3] 1 4 8
```

```
str(factor1)
```

```
##  Factor w/ 3 levels "1","4","8": 1 2 3
```

Merk ook op dat je geen bewerkingen meer kan doen met een numerieke variabele die als factor gebruikt wordt. Dit produceert opnieuw een Error:

```
a*4 # a is geen factor en dit gaat dus wel
```

```
## [1]  4 16 32
```

```
factor1*4 # factor1 is van het data type factor en dit gaat dus niet
```

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

4.8 Logical data type

Een **logical** element kan enkel 2 mogelijke waarden aannemen: *True* of *False*. Dit data type wordt regelmatig gebruikt of verkregen bij het vergelijken van variabelen of het nagaan van logische uitdrukkingen.

```
a <- 5 > 3 # 5 is inderdaad groter dan 3, dus resulteert in 'TRUE'
a
```

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

```
b <- 2*3 == 6

c <- a == b # a en b waren beiden juist en hebben dus beide de waarde 'TRUE'
c
```

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

Merk op dat het testen of twee variabelen gelijk zijn gebeurd met een dubbel gelijkheidsteken '=='. Dit is om verwarring te vermijden met het creëren van variabelen, wat gebeurt met slechts een enkel gelijkheidsteken '='.

Er zijn nog verschillende andere vergelijkingen die gemaakt kunnen worden:

- $<$, $>$, $\leq$, $\geq$ voor ongelijkheden
- == voor gelijkheden
- != voor het testen of 2 elementen of vectoren niet gelijk zijn
- combinaties:
  - &: EN (testen of een uitspraak voldoet aan meer dan 1 voorwaarde)
  - |: OF (testen of een uitspraak voldoet aan minstens 1 voorwaarde)
  - !: NIET (testen of een voorwaarde niet voldoet aan een bepaalde voorwaarde)

Het is belangrijk te vermelden dat bovenstaande operatoren elementsgewijs werken. Dit wil zeggen dat de operator, bij het vergelijken van data types met meerdere elementen, ieder element van een object zal vergelijken met een element op dezelfde positie van een ander element. Bijvoorbeeld, we willen weten welke elementen van twee vectoren aan elkaar gelijk zijn:

```
c1 <- c(1,2,3,4,5)
c2 <- c(2,2,8,9,10)

c1 == c2
```

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

De output vertelt ons dat de elementen van de vectoren verschillen, behalve het tweede element. In het geval dat het doel is om gelijkheid van een vectoren (of andere data types) in hun geheel te testen, kunnen we ook gebruik maken van de functie *identical()*. Deze functie vertelt ons onmiddellijk of de vectoren gelijk zijn, maar geeft geen verdere info over welke elementen verschillen.

```
identical(c1,c2)
```

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

Chapter 5

Een afsluiter

Dit document geeft een korte inleiding tot de programmeertaal van R. Enkele van de vele mogelijke basis functies werden geïllustreerd, maar hier stopt het zeker niet. Er zijn bijvoorbeeld nog meer dan 20 000 R packages die je kan installeren, elk met hun eigen handige functies voor specifieke doelen. Twee handige packages die vaak gebruikt worden en zeker de moeite zijn om eens verder te bekijken, zijn de volgende. Om bijvoorbeeld mooiere grafieken te maken, kan je de functies in het *ggplot2*-package gebruiken. Om makkelijk data sets te bewerken of te filteren, worden de functies in het *dplyr*-package vaak gebruikt.