BASICS OF PROGRAMMING .................................................. P.3

PROGRAMMING WITH R .................................................. P.11

BASICS OF VISUALIZATION .................................................. P.17

VISUALIZATION WITH R .................................................. P.17
A variable is like a box in which we can store data.
A variable is like a box in which we can store data
name the variable as you please

but it must start with a letter

my_variable <- "my data"

DATA TYPES

Text

my_text_variable <- "Hello!"

Variables can store different kinds of data

Number

Boolean
my_number_variable <- 2019

Table

c("B","beta")

my_table[ ,3] <- c("ALPHA","BRAVO")
my_table[2,] <- c("B","beta","bravo")
my_table[2,3] <- "BRAVO"

success <- TRUE
failure <- FALSE
failure <- FALSE
```r
6
my_table <- rbind(c("A","alpha"),
my_table <- cbind(my_table, c("alpha","bravo")))
```
IF - THEN - ELSE

Computations can be executed only under certain conditions

```python
if (age < 18) {
    print("Warning!")
} else {
    print("Welcome")
}
```
IF - THEN - ELSE

Computations can be executed only under certain conditions

if (age < 18) {
    print("Warning!")
} else {
    print("Welcome")
}

test the condition done is passed if test done is failed if test

IF - THEN - ELSE

success <- TRUE
if (success) {
    print("This is executed")
}
if (!success){
The condition is boolean

if(grade == 0){
    print("Terrible Fail!")
} else if(grade < 5){
    print("Fail")
} else if(grade < 8){
    print("Pass")
} else {
    print("Awesome!")
}

Conditions can be tested one after the other

IF - THEN - ELSE

Conditions can be tested together

if(grade > 4 & grade < 5) {
All conditions

print("Almost there!")

if (grade < 2 | grade > 8) {
    print("That’s extreme!")
}

At least one condition

if ( (dish == "Soup" & wine == "None") | (dish == "Fish" & wine != "Red") |
    (dish == "Duck" & wine == "Red") ) {
    print("I want to eat this.")
}

Conditions can be complex and nested

if ( (dish == "Fish" & wine != "Red") &
    (wine == "Rosé" &
     (hour > 23 | temperature > 25) ) ) {
    print("This is an exception.")
}

FUNCTION
A function is a series of operations that can be repeated

my_function <- function(grade){
  if(grade < 5) {
    print("Fail") } else {
    print("Pass") } }

my_function(0)  my_function(8)

FUNCTION

A function is a series of operations that can be repeated

my_function <- function(grade){
  if(grade < 5) {
    print("Fail") } else {

print("Pass") }
my_function(0)
12

Results depend on the input variable
my_function(8)

FUNCTION
give_grade <- function(Q1, Q2){
grade <- 0
if(Q1 == "France") {
grade <- grade + 1
}
if(Q2 == "Louis XIV") {
grade <- grade + 1
}
print(grade)
}

Functions can take several variables as input

Functions can return a single result
test_Q1 <- function(Q1){
if(Q1 == "France") {
}
}
return(TRUE)

Input variables can have default values

grade_Q1 <- function(Q1, grade = 0){
```python
if( test(Q1) ) { } } return(grade + 1)

LOOP

Operations can be repeated a certain number of times

for( i in 1:10 ){
    print(i)
}

for( my_letter in my_table$letter ){
    print(my_letter)
```
Loops can repeat operations until certain conditions change

While Loop

< - 5
while( i > 0 ){
 print( my_table$letter[i] )
 i <- i - 1
}

Loops can be interrupted

for( my_letter in my_table$letter ){
 print(my_letter)
if( my_letter == "B") {
    break
}
}
for( i in 1:10 ){
if( i %in% c(2,4,6,8,10) ) {
    next
} print(i)
}

15

LOOP

Loops can repeat operations until certain conditions change

While Loop

<- 5
while( i > 0 ){
    print( my_table$letter[i] )
    i <- i - 1
}

Loops can be interrupted
for( i in 1:10 ){  
if( i %in% c(2,4,6,8,10) ) {  
    next}
}
for( my_letter in my_table$letter ){  
print(my_letter)  
if( my_letter == "B") {  
    break}
}

Terminate the loop 16

Skip next item
to the

EXERCISE
1. Separate into 2 groups: with and without programming experience

2. Make teams (2-3 max) that mix learners from both groups

3. Download the exercises: link

4. Pitch what was tricky or helpful
R is a programming language

R Studio is an environment for programming, debugging, and executing R.

First install R  https://cran.r-project.org/

Then install R Studio

https://www.rstudio.com/products/rstudio/download/
R Studio has 3 main panels

Find help

20
Using the console on the right side is recommended

Write code
Execute code
Select to execute the lines ... ...press Command of code Enter +

...See the results here!
LIBRARIES

A library is a collection of useful pre-made functions

install.packages("tidyverse")

library(tidyverse)
LIBRARIES

A library is a collection of useful pre-made functions

Download a library
install.packages("tidyverse")
library(tidyverse)

Load a library
...needs no internet ...to do each time
you open R Studio
...needs internet ...to do only once for the whole computer

DATA TYPES

Collections store several elements.
Operations can be applied to all elements.

**Series of Numbers**

my_numbers <- c(3, 5, 7, 11, 13, 17)
my_numbers[7] <- 19
my_numbers <- 1:10

**Series of Texts**

my_texts <- c("Hi", "Salut", "Ciao")
my_texts[4] <- "Hallo"

my_texts <- paste(my_texts, "Mary")

**Series of Booleans**

my_numbers <- my_numbers / 2
myBOOLEANS <- my_numbers == "Hi Mary"
DATA TYPES

Collections can be searched and ordered.

Check the presence of an element:

```r
my_boolean <- "Hi Mary" %in% my_texts
```

Extract unique elements:

```r
if(! "Hi Mary" %in% my_texts ){
  print("'Hi Mary' is missing.")
}
```

Order elements:

```r
sort(my_texts)
sort(1:10, decreasing = TRUE)
```

```r
my_data <- c("Red","Red","Blue", "Red","Black","Black")
unique(my_data)
```
Data types are easily modified Series of Texts

**Number to Text**

```
my_texts <- as.character(1:10)
my_texts <- paste("Item", 1:10)
```

**Text to Number**

```
my_number <- as.numeric("1")
26
my_texts <- c("Hi", "Salut", "Ciao")
```

**Series of Booleans**

```
my_texts <- paste(my_texts, "Mary")
```
my_booleans <- Mary
my_texts == "Hi
my_texts[4] <- "Hallo"

**DATA TYPES**

Dates and timestamps are special data types.

**Date Date & Time**

timestamp <- as.POSIXct("2019-03-07 12:00:00")
now <- Sys.time()
next_minute <- Sys.time() + 60

https://www.r-bloggers.com/date-formats-in-r/
http://www.noamross.net/blog/2014/2/10/using-times-and-dates-in-r---presentation-code.html

date <- as.Date("2019-03-07")
today <- Sys.Date()
tomorrow <- today + 1
format(today, "Year is %Y")
as.numeric(today - tomorrow)
27
DATA TYPES

Variables can store complex data structures

**List**

my_list <- list(my_table, my_texts, my_numbers, "etc")

my_list[[4]] <- "anything"

my_list[[1]][,3] <- c("Alpha","Bravo")

28

**Tibble**

my_tibble <- tibble(1:2, c("a","b"), list(my_texts, my_numbers), list(my_table, my_list))

my_tibble[,2] <- c("A","B")
Data Frame

my_tibble[1,3][[1]][[1]]
[5] <- "Hola"

my_table$code <-
c("alpha", "bravo")

my_data_frame <-
data.frame(my_table)

colnames(my_table) <- c("letter",

PIPE OPERATOR

The pipe ( %>% ) makes our code easier to read

second_function( first_function(data) )
data %>% first_function() %>%
second_function()

...Same as...

PIPE OPERATOR

The pipe ( %>% ) makes our code easier to read

second_function( first_function(data) )
data %>% first_function() %>%

second_function()

This is executed first...

...But we read it last!
Functions take the piped variable as their first input variable.

```
my_function(my_data, my_variable)
```

The pipe can be used anywhere.

```
if( grade %>%
    my_function() > 5 ) {
    print("Pass")
}
```

```
my_function(
)
```

```
grade %>%
round(digits = my_data)
```

```
grade %>%
round(digits = 1)
```
grade %>% round() %>%

my_variable = time %>% round()

Start **TIPS** your scripts with nice settings Cheatsheet
https://www.rstudio.com/resources/cheatsheets/
https://github.com/DigitalSocietySchool/R_FirstContact/tree/master/Cheatsheet

Tutorial
https://r4ds.had.co.nz/
https://www.datacamp.com/tracks/tidyverse-fundamentals
https://www.tidyverse.org/learn/

# Set the working environment if(!require('rstudioapi')) {
install.packages('rstudioapi')
library(rstudioapi)
setwd(dirname(getSourceEditorContext()$path))
# Disable scientific notation
options(scipen = 999)
# Disable text encoding as 'factors'
options(stringsAsFactors = FALSE)


> TEXT FILE

> CSV FILE
READING & WRITING DATA

Format the content of tables TEXT FILE
Write

write("Next Line", "my_text.txt",
append = TRUE)
for( i in 1:nrow(my_table) ){
  paste(
    my_table$name[i],
    "has grade",
    my_table$grade[i] )
} %>% write(
  "class_grade.txt",
} append = TRUE )

Read

read_file("my_text.txt"
)
write("My text",
"my_text.txt")

Use the raw console output
my_table %>%
print() %>%
capture.output() %>%
write("my_text.txt")

CSV FILE

Write as plain text Write tables as they

Read
read.csv("my_data.csv")
write("A, alpha\nB, beta",
"my_data.csv")
write("C, gamma",
"my_data.csv",
append = TRUE)

Use other separator
read.csv("my_data.tsv",
sep = "\t")
EXCEL FILE

Main Libraries
https://www.datacamp.com/community/tutorials/r-tutorial-
Excel files have **different formats** `read_excel` into-R
depending on the version of Excel

*library*(readxl)
*library*(gdata)

The **R libraries** for reading Excel files

*not work with all formats* `library`(xlReadWrite)
*library*(XLConnect)

The right library can be found `library`(xlsx)

with **trials and errors**

36

XML DATA

Main Libraries
XML data has a standard format that also applies to HTML data. The R libraries for reading XML files may work slightly differently. The first step to begin with is to follow a tutorial.

```r
library(xml2)
read_xml("<client>
  <name>Emma</name>
</client>")
```

```r
library(XML)
xmlTreeParse('<client name="Emma">')
```

To access webpages:
library(RCurl)
getURL("https://digitalsocietyschool.org")

library(rvest)
"https://digitalsocietyschool.org" %>%
  read_html() %>%
  html_nodes("a") %>%
  html_attr("href")

library(rvest)
"http://w3schools.com/html/html_tables.asp"
read_html()
read_html()
https://dplyr.tidyverse.org/

- OVERVIEW DATA
- ORDER DATA
- EXTRACT DATA
- ADD DATA
- MERGE DATA
library(tidyverse)

# Get mean & sum of each column
Simple functions for quick checking the data

data %>% colMeans()
data %>% colSums()
# Get summary statistics of each column
data %>% summary()

# Get specific statistics of any column
min(my_column), # Get row names & first
max(my_column) )

# Get statistics for groups of row
# Get first & last rows
summarise( min(my_column), data %>% tail()
max(my_column) )

ORDE

ORDER

DATA

Order by column value

# From lowest to highest column value
data %>% arrange(my_column)  

# From highest to lowest column value  
data %>% arrange(desc(my_column))

# Order by one column then another  
data %>% arrange(my_column, other_column)

41  

Group by column value  

EXTRA
# Get certain columns
```r
data %>% select(my_column, my_column > 5 & my_column < 8)
```

# Get rows with the 10 highest values
```r
data %>% top_n(10, my_column)
```

# Get certain columns
```r
data %>% select(my_column
```

# Random sample 10 rows
```r
data %>% sample_n(10)
```

# Get rows with certain column values
```r
data %>% filter(my_column > 5)
```

# Get rows with certain column values
```r
data %>% filter(my_column > 5, other_column == 0)
```

# Get rows with certain column values
```r
data %>% filter(my_column > 5, my_column < 8)
```

# Get rows with certain column values
```r
https://dplyr.tidyverse.org/reference/filter.html
https://dplyr.tidyverse.org/reference/select.html
```
data %>% select(contains("other"))  
# Get rows with the 10 lowest values

data %>% top_n(-10, my_column)

Add Column

# Add a new column
mutate( new_column = my_numbers )

# Make a column from another column

mutate( bonus = my_column + 2 )

ADD DATA

https://dplyr.tidyverse.org/reference/mutate.html
```
bind_rows(
  my_new_rows
)

Equivalent

left_join(
  my_data, other_data,
  by = c('id_column' = 'other_id_column')
)

right_join(
  other_data, my_data,
  by =
  c('other_column' = 'my_column')
)

Add All Columns

Columns
```
Match on several columns **Keep Only Matching Rows**

```r
inner_join(my_data, other_data,
by = c('my_column'='other_column'))
```
by = c('my_column'='other_column', 'my_extra'='other_extra', )

▶ VISUAL DIMENSIONS ........................................ P.4
▶ MULTIPLE DIMENSIONS ...................................... P.6
▶ NETWORK & FLOW ............................................ P.9
BASICS OF VISUALIZATION

VISUAL DIMENSIONS

Visual features can be used to represent different variables

Length
Variables are the **data dimensions** encoded into **visual dimensions**

Apr.
May
June
July
Aug.
Sept.
Oct.
Nov.
0 50 100 (2 dimensions) (3 dimensions)

**Color**

Apr.
May
June
July
Aug.
Sept.
Oct.
Nov.
0 50 100

**Position**

Apr.
May
June
July
Aug.
Sept.
Oct.
Nov.
0 0 50 50 100 100

0 0 50 50 100 100 (2 dimensions)
Visual features offer many options to represent the same data...

**Area**

100
...but some visual features are hard to compare with human eyes

Length
Area
Position

MULTIPLE DIMENSIONS
To add a data dimension, add a visual dimension

1 Dimension
...but the graph becomes complex

2 Dimensions
3 Dimensions
4 Dimensions
5 Dimensions

MULTIPLE DIMENSIONS

Multiple Views
2014 2015 2016 2017 2018

Apr.
May
June
July
Heatmap
Scatterplot Matrix
MULTIPLE DIMENSIONS
Radar Chart
Cyclic Graph
Radial Graph
Glyph
NETWORK & FLOW

Sankey Diagram Chord Diagram
NETWORK & FLOW

Force-Directed Graph
Tree

Radial Layout
Radial Layout
SAVING GRAPHS

data %>%
ggplot( aes( x = my_column, y = other_column ) ) +
geom_point( aes( text = paste("Tip Box:", another_column) ) ) +
geom_smooth( aes( colour = last_column, fill = last_column ) ) +
facet_wrap( ~ last_column )
ggsave( "my_visualization.pdf", width=18, height=18)
ggsave( "my_image.png", width=10, height=10, unit="cm" )