```r
# Description -------------------------------------------------------------

# In this file, we have a look at some of the most basic concepts of
# programming: Variables and their types.
# You also see how it works to write code in a file (which you should always do).


# Header ------------------------------------------------------------------


# install.packages("tidyverse")
library(tidyverse)

# Whenever we reference to "the book" below, this means
# Wickham and Grolemund, "R for Data Science". Note that
# in the printed version, the Chapter numbers differ from
# the online version on http://r4ds.had.co.nz/.
# Usually, the online numbering is obtained by adding 2 to
# the print version, i.e. the online number is higher.


# Variables and their types -----------------------------

a = 2
a

# See how a appears in the Environment pane?

(b = 5)

# The R community usually uses
a<-2
# This makes it clear that a is ASSIGNED the value 2

# Variables have types. Let's see what's
# the type of a...
typeof(a)
class(a)


# Are there other types?
aAsChar = "a"
aAsChar
## [1] "a"
typeof(aAsChar)
class(aAsChar)

# What is the difference?
a = 2
a == 2

# To make the difference clearer,
# many R programmers use
a <- 2
```

```r
# what is this?
(h = a == 2)

# or this?
(k = a !=aAsChar)

typeof(h)
class(h)



# Combining single objects: Shopping Lists (1) -------------------------



shopList = c("Milk (1)", "Cereals (packs)",
             "Qu\u00f6llfrisch (packs)")



typeof(shopList)
class(shopList)


# Note, I included a so-called UTF-8 character:
"\u00f6"
# Check whether on your machine "ö" would work:
"ö"

# The problem is that it may work on your machine,
# but look weird on someone else's machine. So, to be
# save, use UTF-8 encoding when using non-english letters.

# Other examples:

c("\u00bc", "\u2135", "\u0024", "\u20ac")

# Just google the encoding you need (e.g. "utf8 euro sign")



# shopList is a vector. A vector is
# the basic unit of a variable in R
# even the variable a above was a vector
# (of length 1)

# Check the length of shopList
length(shopList)


# Let's create some other vectors.

quantList = c(3, 2, 5)
```

```r
priceList = c(1.95, 2.05, 11.95)

shoppingData = tibble(shopList, quantList, priceList)

View(shoppingData)
# Instead of coding "View()" you can also just double-click
# on the object name under "Data" in the upper-right pane of RStudio.

typeof(shoppingData)
class(shoppingData)

# Tibble is a relatively recent data type in R that
# is (for the moment being), the best R offers for the
# representation of a "spreadsheet". The tibble format
# is not contained in the R baseline version but comes
# with the tidyverse package (in particular, with dplyr).
# Before these packages were available, the respective
# data type was a "data frame". Data frames still exist
# as data types in the base version of R. For some
# tasks, they are still useful. They are very similar to
# tibbles but have some properties that are sometimes
# annoying.


# Add new info
mutate(shoppingData,
       expPerItem = quantList * priceList
)

shoppingData
#Hm, where is the new info?


# If we want to include the new variable in the
# object shoppingData, we need to assign the updated
# object to that name!

shoppingData = mutate(shoppingData,
       expPerItem = quantList * priceList
)
shoppingData

# Below, we will see an alternative and very elegant
# way to do this using the "pipe". But first something
# else...


# Two ways of getting rid of a colum
D1 = select(shoppingData, shopList, quantList, priceList)

D2 = select(shoppingData, -expPerItem)
# This is far more elegant if you drop just a few columns!
```

```r
# Now we also apply this to the shoppingData
shoppingData = select(shoppingData, -expPerItem)



# Now, we add again the expPerItem column,
# this time with the elegant "pipe":

shoppingDataAug <- shoppingData %>%
  mutate(expPerItem = quantList * priceList)

shoppingDataAug

# Read about the pipe in the book on p. 59 and in Chapter 14.



# What is the difference to the below?
shoppingData %>%
  mutate(expPerItem = quantList * priceList)


# The pipe allows you to chain multiple steps
# without specifying the arguments each time.
# This is why it is so convenient. (If you do just one
# thing, it is not necessarily more convenient than without pipe.)
# E.g.

shoppingData <- shoppingData %>%
  mutate(expPerItem = quantList * priceList,
         expPerItem2 = expPerItem^2)  %>%
  filter(priceList < 10)

shoppingData

# This does the same as

shoppingData = mutate(shoppingData, expPerItem = quantList * priceList)
shoppingData = mutate(shoppingData, expPerItem2 = expPerItem^2)
shoppingData = filter(shoppingData, priceList < 10)

# Do you understand the difference?


# Lists ------------------------------------------------------------------

# A list is a general-purpose container that can contain any
# number and type of elements. For instance, a list can
# also contain a list...

# Here is an example

myList = list(
  firstEl = shoppingData,
```

```
  secondEl =c(1:10),
  thirdEl = rep(0,5),
  fourthEl = seq(from = 2, to = 5.75, by = 0.25),
  fifthEl = shoppingDataAug
)

#You can refer to the elements of the list by using
#<List name>$<element name>. E.g.

myList$secondEl
myList$thirdEl

# Actually, even a dataframe is a list, but one
# that the R developers have given some special
# properties. So it is a "specialized list".

# Check out this:

shoppingData$shopList
shoppingData$quantList

# There is a different way to refer to elements of a list
# (and dataframe) that is sometimes really useful:

shoppingData[["shopList"]]
shoppingData[["quantList"]]

# We will use this when constructing a function
# in 2_DataBasics.

# Here is how you can use the pipe

shoppingData %>%
  .$shopList



# READ ABOUT
#  - Vectors and other data types in the book in Chapter 16
#  - Manipulating tibbles in the book in Chapter 3
#  - Pipes in Chapter 14
```