1 SCR live coding Sept 7, 2019

1.1 Session 1: Objects and Functions

1.1.1 objects a.k.a. variables

```
a <- 9
b <- TRUE
c <- "two"
d <- UNKNOWN # does not work: R has some built-in 'objects', such as numbers, TRUE and FALSE
```

```
## Error in eval(expr, envir, enclos): object 'UNKNOWN' not found
```

```
# notice the colouring in an IDE environment, also e.g. tabbing
```

1.1.2 using functions (and operators)

```
sqrt(a)
```

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

```
print(b)
```

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

`print(b)` is the same as simply entering 'b' in your console

Let's take a look at

```
b * 2
```

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

Apparantly R has some built-in (default) behaviours, TRUE somehow becomes a 1. How about FALSE?

```
d <- FALSE
d * 2
```

```
## [1] 0
```

Usually these default behaviours are convenient, sometimes confusing! Let's get Back to the lecture...

1.2 Session 2: Objects and Functions

1.2.1 creating your own function

```
square <- function(x){
  squared <- x * x
  return(squared)
}
```

Did you see we use the assignment arrow? Actually, functions are also variables / objects. Because we (usually) use them functions in such a fundamentally different way, especially for now, we do not view them as variables or objects.

```
square(a)
```

```
## [1] 81
```

```
sqrt(square(a))
```

```
## [1] 9
```

```
square(b) # error!
```

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

1.2.2 side effects and errors.. NA / NaN / Null?

About `Na`

```
as.numeric("four")
```

```
## Warning: NAs introduced by coercion
```

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

```
as.numeric(c("3", "2", "...", "1"))
```

```
## Warning: NAs introduced by coercion
```

```
## [1]  3  2 NA  1
```

About `NaN`

```
1/0 - 1/0
c(1, 2, NA, 4)
```

About `NULL`

```
a <- plot("")
c(3, 2, NULL, 1)
```

Some checks:

```
is.na(NA)
```

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

```
is.null(NULL)
```

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

```
is.null(NA)
```

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

and tricky ones:

```
is.na(NULL) # the answer is a logical vector (TRUE / FALSE) of length 0. E.g. an empty vector
```

```
## Warning in is.na(NULL): is.na() applied to non-(list or vector) of type
## 'NULL'
```

```
## logical(0)
```

```
is.null(logical(0)) # an empty vector != nothingness
```

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

Meanwhile on your hard drive:

```
object.size(NULL)
```

```
## 0 bytes
```

```
object.size(NA)
```

```
## 48 bytes
```

Some functions have an inbuild option to remove NA values:

```
x <- c(1, 2, NA, 4)
var(x)
```

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

```
var(x, na.rm = TRUE)
```

```
## [1] 2.333333
```

Let's get back to the lecture slides

1.3 (Long) Session 3: Vectors

1.3.1 Type/ Mode

R automatically decides the mode (what type of thing the container contains), often easy:

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

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

```
class(a)
```

```
## [1] "numeric"
```

```
typeof(a)
```

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

```
b <- c("a", "bunch", "of", "strings")
b
```

```
## [1] "a"       "bunch"   "of"      "strings"
```

```
typeof(b)
```

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

```
c <- c(TRUE, FALSE, T, T, F) # TRUE is same as T, FALSE is same as F
c
```

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

```
typeof(c)
```

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

and sometimes hard:

```
d <- c(4, "word", T)
typeof(d)
```

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

```
d
```

```
## [1] "4"    "word" "TRUE"
```

we can force behaviours using as.<*> functions:

```
as.numeric(d)
```

```
## Warning: NAs introduced by coercion
```

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

```
as.logical(d)
```

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

Sometimes, conviently, R knows how to interpret text.

```
as.numeric(c("1", "two", "3", 4))
```

```
## Warning: NAs introduced by coercion
```

```
## [1]  1 NA  3  4
```

but R does not 'know' english, and does recognize numbers!Last, note the difference between warnings and errors! Here we get warnings, R does not break down.

1.3.2 Object properties:

objects have properties, e.g. a vector has a length:

```
length(a)
```

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

you can access a particular element using `[]`

```
a[1]
```

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

each element in a vector can be given a name

```
names(a) # NULL first!
```

```
## NULL
```

```
names(a) <- b
a
```

```
##       a   bunch      of strings
##       1       2       3       4
```

Some useful ways of making vectors:

```
e <- character(10) # creates an 'empty' vector of type character, with space reserved for 100 elements
e[5] <- "the fifth entry"
```

```
f <- 1:10
g <- -5:3
h <- seq(0, 1, by = 0.1)
```

```
class(f)
```

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

```
class(h)
```

```
## [1] "numeric"
```

actually there are multiple 'vector' classes: for each type of mode, a unique vector class exists (but all containing just one mode).

There is another property called 'dim': the dimensionality of the vector

```
class(f)
```

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

```
dim(f)
```

```
## NULL
```

```
dim(f) <- c(2, 5)
class(f)
```

```
## [1] "matrix"
```

1.3.3 Vectorized Functions

Some functions can be applied to vectors:

```
a * 2
```

```
##       a   bunch      of strings
##       2       4       6       8
```

```
a
```

```
##       a   bunch      of strings
##       1       2       3       4
```

```
a * a
```

```
##       a   bunch      of strings
##       1       4       9      16
```

```
g <- 1:3
g * 3
```

```
## [1] 3 6 9
```

```
1:3 * 2
```

```
## [1] 2 4 6
```

We called this 'vectorized' functions, they are usually superfast.

Some functions require vectors:

```
mean(a)
```

```
## [1] 2.5
```

Some functions behaviour depends on whether argument is vector or not:

```
diag(3) # diagonal matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
diag(c(1, 2, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

The behaviour of functions in R depends on the arguments you give it. R uses the information about the argument (the class) to determine which behaviour to perform! Be aware! Often only a single class may be used.

1.3.4 IMPORTANT: Quirks and Perks: Vector Recycling!

```
a <- 1:2
b <- 1:3
a * b
```

```
## Warning in a * b: longer object length is not a multiple of shorter object
## length
```

```
## [1] 1 4 3
```

Note how the first element of `a` is used for a second time to multiply it with the 3rd element in `b`!

1.4 Session 4:

1.4.1 Indexing:

Let us create a simple vector as an example

```
x <- 1:10
x
```

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

Let us give each of the entries a name

```
names(x) <- letters[1:10]
x
```

```
x[1]
```

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

```
x["b"]
```

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

```
x[-1]
```

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

```
x[c(T, F, F, F, F, F, F, F, F, F)]
```

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

What happens if we do:

```
x[0]
```

```
## integer(0)
```

```
x[11]
```

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

```
x["m"]
```

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

Mind blowing! The Expansive power of vector recycling:

```
rep(c(1,2), each = 10)
```

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

```
Booleans <- c(T,F)
Booleans[rep(c(1,2), each = 10)]
```

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

1.4.2 filtering

Using logical operators we can make R check if a particular condition is TRUE or FALSE

```
1 < 2
```

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

```
2 < 1
```

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

Using logical operators applied to vectors we can automatically created vectors with corresponding TRUE and FALSE that check, elementwise, if the condition holds for the elements in x

```
x > 5
```

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

```
x[x > 5]
```

```
## [1]  6  7  8  9 10
```

1.5 Session 5:

1.5.1 Workspace

Usually your script is enough to reproduce an entire workspace

Sometimes analyses will take a very long time and saving the script only can be very inconvenient

```
ls()
```

```
##  [1] "a"        "b"        "Booleans" "c"        "d"        "e"
##  [7] "f"        "g"        "h"        "square"   "x"
```

```
save(list=ls(), file="my_workspace.Rdata") # this stores your workspace 'somewhere' on your PC, we'll l
rm(list=ls()) # remove everything
# now let's load everything back in
load("my_workspace.Rdata")
```

If you really want to save everything you've ever done:

```
savehistory(file = ".Rhistory")
```

1.5.2 Packages:

base is a package loaded into R by default

```
library("base")
```

R won't give you any feedback unless something may have gone wrong. For anay help on a library, or function:

```
library(help="base")
?sample
```

a typical library you might use that is not loaded by default, but installed by default

```
library("foreign")
library("MASS")
```

a new library that you will need to use for extra exercises at 'home'

```
install.packages("swirl")
library("swirl")
rm(list=ls())
swirl()
```