Session 1: Getting to Know R

QM-1

2022-10-10

R as a calculator

```
4*10^1 + 2*10^0 + 1*10^(-1) + 4*10^(-2)
```

```
## [1] 42.14
```

```
1/100000000
```

```
## [1] 1e-08
```

```
2.34e-02
```

```
## [1] 0.0234
```

```
13/11  # a cutoff after 6 digits (per default)
```

```
## [1] 1.181818
```

```
print(13/11, digits = 9) # display 9 digits
```

```
## [1] 1.18181818
```

```
sqrt(2)
```

```
## [1] 1.414214
```

```
2^.5
```

```
## [1] 1.414214
```

```
pi
```

```
## [1] 3.141593
```

```
27^(1/3)
```

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

```
(1/32)^(1/5)
```

```
## [1] 0.5
```

```
.0001^.25 #no need for leading zeros in front of '.'
```

```
## [1] 0.1
```

Some more built in functions

```
exp(1) # the Euler constant
```

```
## [1] 2.718282
```

```
exp(5)
```

```
## [1] 148.4132
```

```
log(exp(5))
```

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

```
choose(5, 2) # calculates the binomial coefficient.
```

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

Getting help in R:

```
#How can I get help with built in functions?
?log
```

```
## starting httpd help server ... done
```

```
?choose

#If the name of the command is unknown, typing two question marks activates a buzzword search.
?? '!='
```

Storing values a.k.a. the "gets" operator:

```
r <- 4
13/11 -> somenumber
1.02*r -> r # the "old" r will be overwritten

points <- 30 * (1 + .8 + 1.1)/3
points
```

```
## [1] 29
```

```
(points <- 30 * (1 + .8 + 1.1) / 3) #result is displayed
```

```
## [1] 29
```

```
# R is  vectorized.
```

In order to create a vector use c(...) - for combine

```
r <- c(2,3,4,5) # a vector with the entries  2, 3,4,5 is generated

2:5   #the same result can be achieved using colon operator
```

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

```
1:3
```

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

```
r^2  # squares all entries of the vector elementwise
```

```
## [1]  4  9 16 25
```

```
r2 <- 1:4
r*r2
```

```
## [1]  2  6 12 20
```

```
r+r2
```

```
## [1] 3 5 7 9
```

```
r-r2
```

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

```
r/r2
```

```
## [1] 2.000000 1.500000 1.333333 1.250000
```

```
r+5
```

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

```
r*5
```

```
## [1] 10 15 20 25
```

```
r-5
```

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

```
r/5
```

```
## [1] 0.4 0.6 0.8 1.0
```

```
r3 <- 1:3
r+r3 # recycling in the shorter vector
```

```
## Warning in r + r3: longer object length is not a multiple of shorter object
## length
```

```
## [1] 3 5 7 6
```

```
rsomething <- c(r,r) # combine two vectors to one new vector
rsomething
```

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

More complex sequences (seq, rep, ...):

```
seq(1, 10, by = 2) #sequence from 1 to 10 with step size 2
```

```
## [1] 1 3 5 7 9
```

```
seq(10,1, by=-1) #sequence from 10 to 1 with step size -1
```

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

```
rep(r, 2) #replicates the vector 2 times
```

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

```
rep(r, 2, each = 2) #replicates the vector 2 times and each entry 2 times
```

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

Even more built in functions (length, sum, min, summary, mean, prod, ...):

```
mean(r)
```

```
## [1] 3.5
```

```
min(r)
```

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

```
length(r)
```

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

```
sum(r)
```

```
## [1] 14
```

Simple indexing:

```
r[4]
```

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

```
r[c(1,3)]
```

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

```
r[-4]
```

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

```
r[-c(1,3)]
```

```
## [1] 3 5
```

Comparison Operators

```
2 == 3  #== sign use to compare the left and right handside
```

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

```
2 < 3
```

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

```
r < 4
```

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

```
1 != 1  # != ... used to ask/check for unequality
```

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

```
r < 5 & r > 2   # logical 'and' operator works elementwise in a vector
```

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

```
r < 3 | r > 3 # logical 'or' workds elementwise in a vector
```

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

WARNING: && and || both work as the logical 'and' and the logical 'or', respectively, too, but they are not vectorized!

```
r < 5 && r > 2 # only the first element is checked
```

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

```
r < 3 || r > 3 # only the first element is checked
```

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

More complex indexing:

```
r[r < 4]
```

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

```
r[r != 4]
```

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