STAT 513/413: Lecture 3
R in style and spirit

(looks are important)

One reason STAT 513 was created

Last time, we arrived to a script that looked like this

```
m=1
n=0
for (k in 1:20) { m[k]=k
n[k]=2+3*m[k]+rnorm(1)
plot(m,n)
```


What is wrong with that?

A simple answer

Does any published book with R feature a code like that? Really, does it?

Most of the code out there is typically:

- monospaced
- properly styled

and also

- in the R spirit
- and sometimes commented

(On the other hand: nothing is a dogma here, and there is almost always more than one way to do it)

But in this course we better agree on some standards

So let us work on a code improvement

"Monospaced" is easy

Just use the appropriate font, or even better, the appropriate editor, or even better, the appropriate format

```
m=1
n=0
for (k in 1:20) { m[k]=k
n[k]=2+3*m[k]+rnorm(1) }
plot(m,n)
```

Now: style

Well, there is a "bylaw" on that, but roughly this:

- code inside braces should be indented
- indent is two or four spaces (consistently throughout though)
- ...unless you continue a function: then you return where it started
- because you should break long lines into nicer shorter ones
- closing brace `}` should have its own line
- there should be spaces...
- ...but not excessively many of them (no `function ( x , y ) `, say)

Some refined ones:

- use `<-`, not `=`, and certainly not `->`
- use `TRUE`, `FALSE`, not merely `T`, `F`

Finally, comments: you should use, but not abuse; use taste

References on the "bylaw"

More precisely here:

`https://style.tidyverse.org`

`http://adv-r.had.co.nz/Style.html`

`https://google.github.io/styleguide/Rguide.xml`

R Code Style R Blogs (RStudio)

And the best way to that is...

... via the programming editor does it for you automagically

(note: it is important your files have extension .R)

Some of those are

- ESS with Emacs
- RStudio (configurable), ATOM, ...

It is also possible to run your code through R packages:

- `styler`
- `formatR`

So: organization

```
m=1
n=0
for (k in 1:20) {
  m[k]=k
  n[k]=2+3*m[k]+rnorm(1)
}
plot(m,n)
```


This is a bit "C style"; some may prefer


```
m=1
n=0
for (k in 1:20)
{
  m[k]=k
  n[k]=2+3*m[k]+rnorm(1)
}
plot(m,n)
```

With us, both are fine

Ah, spacing now!

```
m = 1
n = 0
for (k in 1:20) {
  m[k] = k
  n[k] = 2 + 3 * m[k] + rnorm(1)
}
plot(m, n)
```


Here, there is more leeway; I personally prefer less in formulas. Somebody else may add also vertical spaces, to separate important blocks of code:

```
m = 1
n = 0

for (k in 1:20) {
  m[k] = k
  n[k] = 2 + 3*m[k] + rnorm(1)
}

plot(m, n)
```

And let us do also the assignments

Well, at least if you want to publish book on R, you cannot go with "=" ... But on the other hand, you may be also a bit fancy

```
m <- 1; n <- 0

for (k in 1:20) {
  m[k] <- k
  n[k] <- 2 + 3*m[k] + rnorm(1)
}

plot(m, n)
```

So, could I publish the book on R?

(Everybody did already...)

Well, the code *look* is OK now - but the contents

For instance, you do not do loops in R: you vectorize if you can...

Rule of thumb: the less lines of code in R, the better.

But this is cheap:


```
m <- 1; n <- 0
```


Succesful vectorization is much better - how about this


```
m <- 1:20
n <- 2 + 3*m + rnorm(20)
plot(m, n)
```

(There is no need for empty lines - they would not count anyway - when there are only three lines of code altogether)

So, what is the R spirit?

Well, this aspect is not that easily encapsulated into few guidelines - we will rather strive all this course to get an idea what it is

But one thing we may start immediately with:

avoid loops...

... think in terms of vectors/matrices, if possible

Another one, related to the previous one: use the code of experts

For now, perhaps the last touch

```
# points scattered about a line
m <- 1:20
n <- 2 + 3*m + rnorm(20)
plot(m, n)
```

But do not overdo it

Comments yes, but less is more - unlike this

```
# points scattered about a line
# assign 1:20 to m
m <- 1:20
# n lines on the line 2+3m + random error
n <- 2 + 3*m + rnorm(20)
# plotting the result
plot(m, n)
```


If at all - if you really must - then at least like this

```
### points scattered about a line
m <- 1:20                    # uniformly spread
n <- 2 + 3*m + rnorm(20)     # normal error
plot(m, n)
```

Modus operandi already mentioned: functions

• **function**: the input can be varied in a better way than a script - which has to be reedited - and the variables inside the function do not mess up in your working environment (scoping)

```
line <- function(x, s=1, a=2, b=3)
### plots x points approximately following a line with given
### intercept and slope, plus normal error controled by s
{
  m <- 1:x
  n <- a + b*m + rnorm(x,0,s)
  plot(m, n)
}
```


All this process enables you to vary input - first in script, then in function - and thus get some more confidence that the whole concoction does the right thing

However, once again: for this course we are just fine with scripts - although successful scripts can be easily upgraded to functions, and those are allowed as well

However: a word about packages

Packages, add-ons, are very useful at times; they may save us unnecessary work

However, this course is not about R, but about statistical computing. This implies the following rule

PACKAGES ARE NOT TO BE USED

unless (every rule has an exemption) they are not essential to the understanding of what needs to be done

Example: if a problem asks for constructing a generator of random numbers with a prescribed distribution, then its solution is not finding on the internet a package that does it. That misses the point; it is better to learn something via programming it. But, if such a generator is just a small component used for achieving a more complex objective, it is fine to use a package

If in doubt, better ask!