

# Programming with R 2020 University of Potsdam

Detlef Groth  
Summer Semester 2020

KISS:  
“Keep it simple, stupid!”?  
:)  
Paul D. Stroop 1960-12-04

# Which Language?

- Perl 4/5/6 (1987, 1994, 2015) Guru: Larry Wall
  - Python 2/3 (1994, 2008) - Guru: Guido van Rossum
  - S/S3, (1976/1988), R (2000) - Ross Ihaka und Robert Gentleman)
  - Tcl/Tk 1988/1991, Tcl 8.0 1997 - John Ousterhout
- ⇒ in this course we use R
- Here: R as a general purpose programming language!!
  - Normal: R is used in the domain of statistics!!
  - “Programming with R” replaces the old course “Statistics with R”

# R in the Master Bioinformatics

- Statistical Bioinformatics, M1
- (Mathematical and Algorithmical Bioinformatics, M1)
- (Databases and Practical Programming, B1)
- Analysis of Cellular Networks, E2
- Machine Learning in Bioinformatics, E2
- Structural Bioinformatics, E2
- Quantitative Genetics, E3
- (Adv. meth. for Analysis of Biochemical networks, E3)
- Project work, Master thesis

⇒ Python used in 2-3 modules

⇒ Matlab used in 2-3 modules

⇒ C/C++ used in Programming Expertise (B2)

# Hello World!

```
groth:~$ R --slave -e 'print("Hello World!")'  
[1] "Hello World!"
```

```
groth:~$ Rscript -e 'print("Hello World!")'  
[1] "Hello World!"
```

# Outline

## Day 1 (basics)

- setup, install editor, simple programs
- variables, operators
- data structures, control flow

## Day 2 (basics)

- functions
- file input/output
- terminal interaction
- command line arguments

## Day 3 (advanced)

- object oriented programming
- code documentation

## Day 4 (advanced)

- using packages
- writing packages
- package documentation

## Day 5 (advanced)

- graphical user interfaces
- tcltk (shiny)

# Examination

- two short written questions (10 minutes)
- practical programming task for about 90 minutes (3/4 CP)
- for 6 credit points MS-BAM students have to complete a more extensive homework within around a month
- dates:
  - 1.) Thu, October 1st, Review session 1 13:00
  - 2.) Tue, October 6th, Exam 1 13:00
  - 3.) Mon, October 26th, Review session 2 13:00
  - 4.) Wed, October 28th, Exam 2 13:00

# Course Procedure

- video materials for each lecture around 90min
- 10:00-11:30 self study of video materials
- 12:00-13:00 seminar about the lecture (Zoom)
- 13:30-16:00 practical programming using Padup-Chat and Zoom
- if you know one programming language learning the others will be much easier ...



# Lecture Materials / Online Resources

- Videos:
  - Derek Banas YT - R Tutorial
- Links, Books
  - <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>
  - <https://www.tutorialspoint.com/r/>
- More links to tutorials, reference cards on Moodle

# Moodle Course

- Moodle: `https://moodle2.uni-potsdam.de/course/view.php?id=22964`
- Moodle-Login: E-mail account credentials
- Course key: Golm2020

# Outline

<b>Table of Contents</b>	<b>11</b>
<b>1 Intro</b>	<b>16</b>
1.1 R Basics . . . . .	19
1.2 Programming Intro . . . . .	42
1.3 Variables and Data Containers . . . . .	43
1.4 System Variables . . . . .	56
1.5 Help . . . . .	57
1.6 Eval . . . . .	61
1.7 Operators . . . . .	63
1.8 Control Flow: Conditionals . . . . .	65
1.9 Control Flow: Loops . . . . .	70

1.10	Functions . . . . .	75
1.11	Colored Terminal . . . . .	80
1.12	Exercise Introduction . . . . .	86
<b>2</b>	<b>Functions</b>	<b>95</b>
2.1	Definition and code organization . . . . .	97
2.1.1	Definition . . . . .	97
2.1.2	The main function . . . . .	102
2.1.3	Function arguments . . . . .	106
2.2	File IO . . . . .	114
2.3	Command line arguments . . . . .	126
2.4	Terminal applications . . . . .	139
2.5	Exercise Functions . . . . .	154

<b>3</b>	<b>Object Oriented Programming</b>	<b>165</b>
3.1	Object Oriented Programming . . . . .	167
3.2	S3 . . . . .	172
3.3	Proto . . . . .	178
3.4	RDS files . . . . .	189
3.5	Code documentation . . . . .	192
3.6	R graphics system . . . . .	206
3.7	Exercise OOP . . . . .	221
<b>4</b>	<b>R Packages</b>	<b>231</b>
4.1	Installing packages from the web . . . . .	233
4.2	Using packages . . . . .	256
4.3	Writing packages . . . . .	260
4.4	Exercise 4 - packages . . . . .	288

<b>5</b>	<b>R-GUIs with tcltk</b>	<b>300</b>
5.1	Hello World . . . . .	306
5.2	Layout Widgets . . . . .	312
5.3	Basic Widgets . . . . .	318
5.4	Documentation . . . . .	328
5.5	Dialogs . . . . .	333
5.6	Menues . . . . .	343
5.7	Advanced Widgets . . . . .	346
5.8	Advanced Layout Widgets . . . . .	357
5.9	Events . . . . .	363
5.10	Exercise 4 - R-tcltk . . . . .	384

# Programming with R

## SS 2020

### Introduction

Detlef Groth  
2020

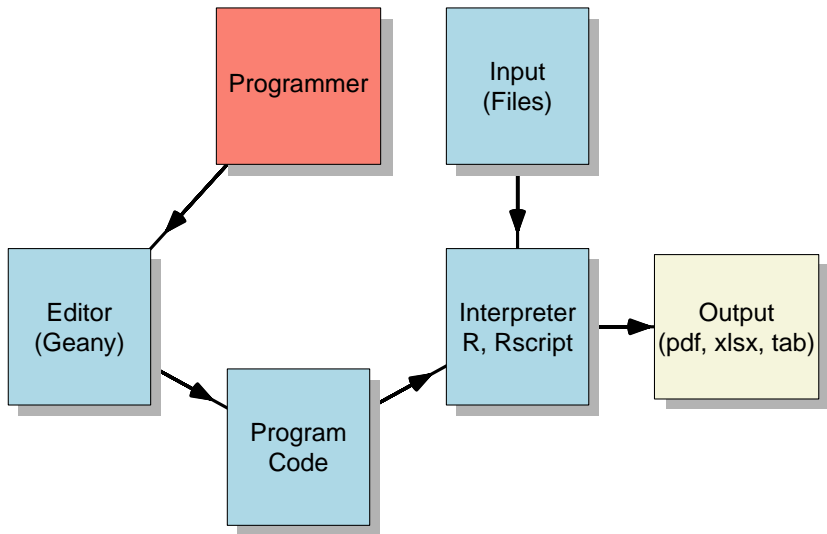
# 1 Intro

## Outline

1.1	R Basics . . . . .	19
1.2	Programming Intro . . . . .	42
1.3	Variables and Data Containers . . . . .	43
1.4	System Variables . . . . .	56
1.5	Help . . . . .	57
1.6	Eval . . . . .	61
1.7	Operators . . . . .	63
1.8	Control Flow: Conditionals . . . . .	65
1.9	Control Flow: Loops . . . . .	70
1.10	Functions . . . . .	75
1.11	Colored Terminal . . . . .	80
1.12	Exercise Introduction . . . . .	86



# Our Workflow



# Software

- OS: recommened Linux (Windows, OSX)
- R programming software environment  
<http://www.r-project.org>
- Text editor Geany <https://www.geany.org/>

# 1.1 R Basics



- R is an implementation of the S-Language.
- S was originally developed at Bell Laboratories in the 1980s, 1990s.
- Implementations (GUI and command line):
- S-Plus on Win32, Unix (commercial Insightful Co), 1995 V 3.3, 1997 first version for Windows.
- R (open source, Win32, Unix, Mac-OSX), 1997 first alpha, 2000 V 1.0, currently 3.6.3.
- We will use the R variant which is free.
- Knowledge validity? 2000 ... 2020 ... 2030 ...

# R-features

- Available for Win32, Linux, Mac-OSX and others OS.
- Mainly console driven, but GUIs are available
- Shell with history, “TAB”-completion of variables, functions, objects as in Unix :)
- Extensible programming language
- Large amount of packages (CRAN)
- Bioconductor project for packages useful to analyse genomics data
- <http://www.bioconductor.org/>
- Sometimes too many packages although ...

# Working with R

- interactive in the R terminal, good for statistics and data exploration
- using script files from the terminal
- in this course we prefer script files as this requires correct code in your files

**We learn to use R without doing Statistics!!**

# Hello World!

```
#!/usr/bin/Rscript  
print("Hello R-World! 2020!") ;  
⇒ [1] "Hello R-World! 2020!"
```



⇒ `#!/usr/bin/Rscript` ⇒ shebang (Unix)

# Version

```
#!/usr/bin/Rscript  
print(R.version.string);  
⇒ [1] "R version 3.6.3 (2020-02-29) "
```

 version.R

# Language vs Interpreter

Language	Interpreter Compiler	Interactive	Tiobe- Index
C/C++	gcc, g++	gdb	1 and 4
Python	python3	python3, idle3	3 (3, 4)
R	R, Rscript	R, RStudio	9 (15, 14)
Perl	perl	perl -de1, pirl	13 (19, 16)
Matlab	matlab	matlab	16 (20, 11)
Julia	julia	julia	28 (42, 37)



# Modus Operandi

- No compilation to machine code necessary with R, Perl, Python.
- How we can work with scripting languages?
  - interactive (R!!, Python3).
  - one-liners in the terminal (Perl!!)
  - **run code from saved source code script files** (R, Perl, Python)
  - for the latter you need a good text editor!

# Text Editor

- edit in plain text mode
- syntax highlighting (R, Python, C, C++, Octave/Matlab, LaTeX, Markdown, ...)
- list of functions, classes overview
- code snippets, templates
- fast and lightweight
- project management (folder based)
- external tools for code snippets (no support by me):
  - Windows: autohotkey
  - Linux (X11): autokey
  - OSX: Hammerspoon

# Why not Rstudio?

- Rstudio great for interactive sessions and learning
- students however are not encouraged to write running applications
- often code mess of different trials instead of structured programs
- does not encourages structured work within different projects
- very memory intensive, often crashing
- may be too many features
- quite R specific, although support for Python and others are improving
- outline of functions not very sophisticated

*Rstudio encourages an unstructured working style  
(DG)*

# Why Geany?

- <https://www.geany.org>
- general purpose text editor
- supports many more languages than Rstudi (Python, R, LaTeX, Perl, Matlab/Octave, C, C++, ...)
- use one editor for all of your programming tasks
- encourages good code and project organization
- teaches you to understand what is going on in the background



## Other editors

- hardcore programmers: Emacs, Vi, Vim
- easier variants: Geany, Kate, Gedit, Atom, Visual Studio Code
- I use MicroEmacs (last update in 2010) ...
- RStudio is good for learning but not for coding! (personal opinion)
- RStudio license is dubious
- we will use **Geany** in this course  
<https://www.geany.org/> like in the statistics course
- Geany can be used with a lot of Programming languages and is available for all the major operating systems

# Installs

- **Linux-Fedora:**

```
sudo dnf install geany R
```

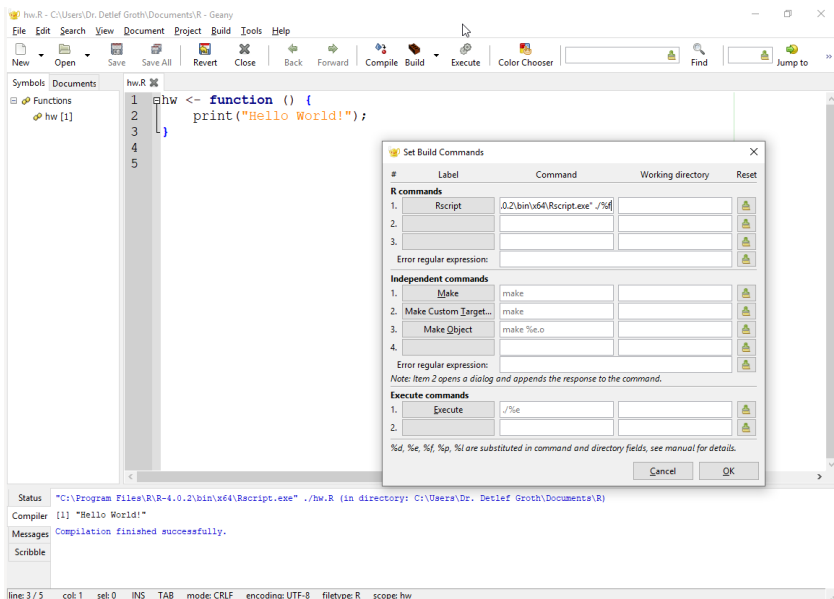
- **Linux-Ubuntu:**

```
sudo apt-get install geany R
```

- **Windows/MacOSX**

- **Geany:** <https://www.geany.org/download/releases/>
- **R:** <https://lib.ugent.be/CRAN/>

# Setup R for Geany on Windows



# Setup R for Geany on Linux

The screenshot shows the Geany IDE interface. The main editor displays an R script with the following code:

```
1 hw <- function () {  
2   print("Hello World!")  
3 }  
4  
5 hw()  
6
```

The 'Symbols' pane on the left shows a function named 'hw' with one argument.

The 'Set Build Commands' dialog is open, showing the following configuration:

#	Label	Command	Working directory	Reset
<b>R commands</b>				
1.	Rscript	Rscript %f		
2.				
3.				
		Error regular expression:		
<b>Independent commands</b>				
1.	Make	make		
2.	Make Custom Target...	make		
3.	Make Object	make %e.o		
4.				
		Error regular expression:		
Note: Item 2 opens a dialog and appends the response to the command.				
<b>Execute commands</b>				
1.	Execute	./%e		
2.				

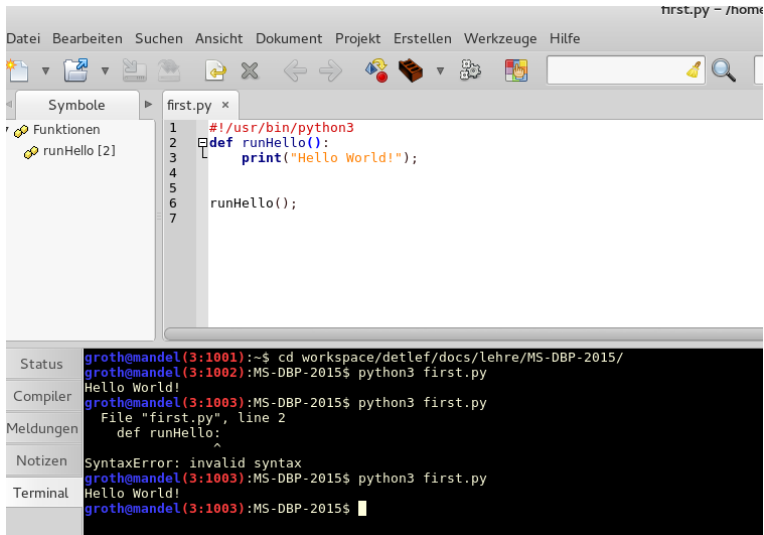
At the bottom of the dialog, it states: "%d, %e, %f, %p, %l are substituted in command and directory fields, see manual for details." Buttons for 'Cancel' and 'OK' are at the bottom right.

The status bar at the bottom left shows the following messages:

```
Status  
[1] "Hello World!"  
Compiler  
Compilation finished successfully.  
Messages
```



# Editor Setup: Geany - Python



# Editor Setup: Geany - Perl

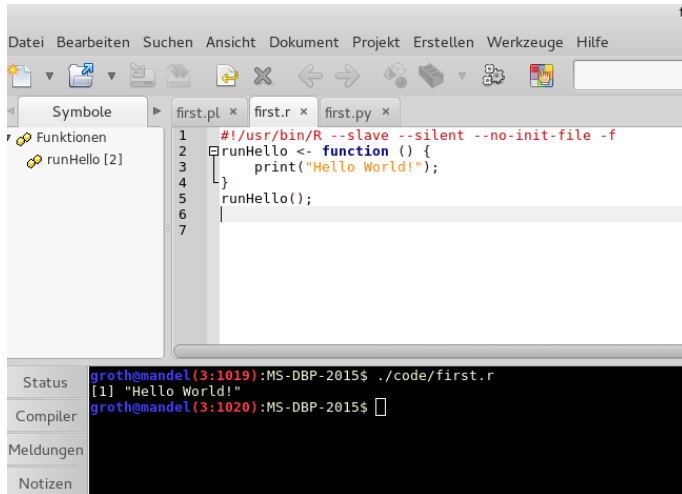
The screenshot displays the Geany IDE interface. The top menu bar includes 'View', 'Open', 'Save', 'Save All', 'Revert', 'Close', 'Back', 'Forward', 'Compile', and 'Build'. The left sidebar shows a 'Symbols' pane with a tree view containing 'Functions' and 'runHello [3]'. The main editor window has three tabs: 'untitled', 'untitled', and 'geany-perl.pl'. The 'geany-perl.pl' tab is active, showing a Perl script with the following code:

```
1  #!/usr/bin/perl
2
3  sub runHello {
4      print("Hello World!\n");
5  }
6
7  runHello();
8
```

Below the editor is a terminal window with a status bar on the left. The status bar shows 'Status', 'Compiler', 'Messages', 'Scribble', and 'Terminal'. The terminal output is as follows:

```
groth@laerche(5:1001):~$ cd ~/workspace/detlef/docs/lehre/MS-DBP-2017/
groth@laerche(5:1002):MS-DBP-2017$ perl code/geany-perl.pl
Hello World!
groth@laerche(5:1003):MS-DBP-2017$
```

# Editor Setup: Geany - R



# Snippets

Geany Wiki: “Snippets are small strings or code constructs which can be replaced or completed to a more complex string. So you can save a lot of time when typing common strings and letting Geany do the work for you.”

- Tools->Configuration Files->snippets.conf
- `https://www.geany.org/manual/current/index.html#user-definable-snippets`
- `https://wiki.geany.org/snippets/start`

# Standalone Scripts: Shebang, Linux, OSX

- shebang: first line of a file starts with `#!` and the path to a script interpreter
- rest of the file is run with this interpreter
- use  
`chmod 755 filename`  
to make a file executable

```
#!/bin/sh
# next line is executed by /bin/sh
echo 'Hello World!'
```

# Standalone Scripts: chmod 755 / Linux

```
groth@mandel:~$ cat ./code/first.py
```

```
#!/usr/bin/python3
```

```
def runHello():
```

```
    print('Hello World!');
```

```
runHello();
```

```
groth@mandel:~$ ./code/first.py
```

```
bash: ./code/first.py: Keine Berechtigung
```

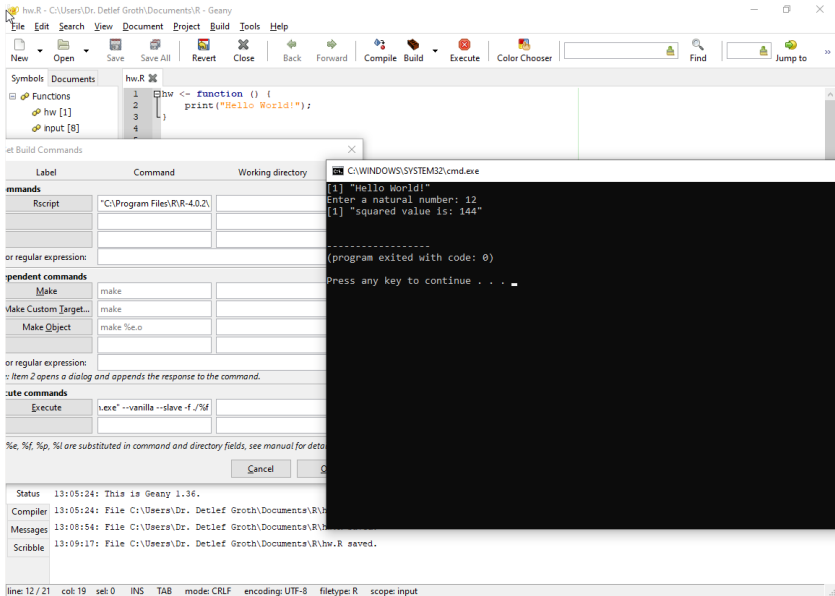
```
groth@mandel:~$ chmod 755 ./code/first.py
```

```
groth@mandel:~$ ./code/first.py
```

```
Hello World!
```

```
groth@mandel:~$ cat ./code/first.r
#!/usr/bin/R --slave --no-init-file -f
runHello <- function () {
    print('Hello World!');
}
runHello();
groth@mandel:~$ chmod 755 ./code/first.r
groth@mandel:~$ ./code/first.r
[1] 'Hello World!'
```

# Geany Windows with execute





# Windows Paths with Geany

Build->Set- Build Comands -> Execute

```
"C:\Program Files\R\R-4.0.2\bin\x64\Rterm.exe"  
--vanilla --slave -f %f
```

# 1.2 Programming Intro

## Concepts:

- variables
- data types
- data containers
- control flow
- functions - day 2
- objects, classes, day 3
- libraries/packages, day 4
- user! interfaces, day 5 (writing programs for others)

## 1.3 Variables and Data Containers

- variable: one or more values
- container: structure to store those values
- reserve some space in memory for the values
- different data types (different space requirements):
  - numbers (integer, long, float, double)
  - strings
  - booleans
  - factors (classified strings)
- some basic data structures:
  - skalar: single values
  - vector, array (matrix): many values, same type
  - list, data frame: many values, diff. types possible
  - others: tibble, table, graph, sql-table

## Variables:

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to the variables, you can store integers, decimals or characters in these variables.

[Tutorialspoint — Python 3 – Variables, 2019]

- variables are an alias for data or other objects (functions, database connection, ...)
- we have always **global and local scope** of variables
- local scope means restricted access, for instance inside functions or objects
- global scope means visible everywhere in your program
- hint: it is advisable to avoid global scope
- we have some basic data types for (data) variables
- R data types:
  - numeric
  - string
  - boolean
  - factor
  - ...

# Variables: R

- local scope assignment operator:
  - `var = value`
  - `var <- value`
  - `value -> var`
- all the same: `x = 3; x <- 3; 3 -> x`
- if used at the global namespace they are global
- global scope assignment operator `<<-` inside a function:  
`global.x <<- 3`
- hidden variables: start with a dot `.hidden=1`
- command `ls()` lists all variables but not hidden ones

# Variables: R-Session

```
> global.x=3
> .hidden.var = 1
> test.func = function () { y = 3 ; z <<- 4 ;
+     global.x <<- global.x +1
+ }
> ls()
[1] "global.x"    "test.func"
> test.func()
> ls()
[1] "global.x"    "test.func"   "z"
> global.x
[1] 4
> test.func()
> global.x
[1] 5
```

# Summary variables

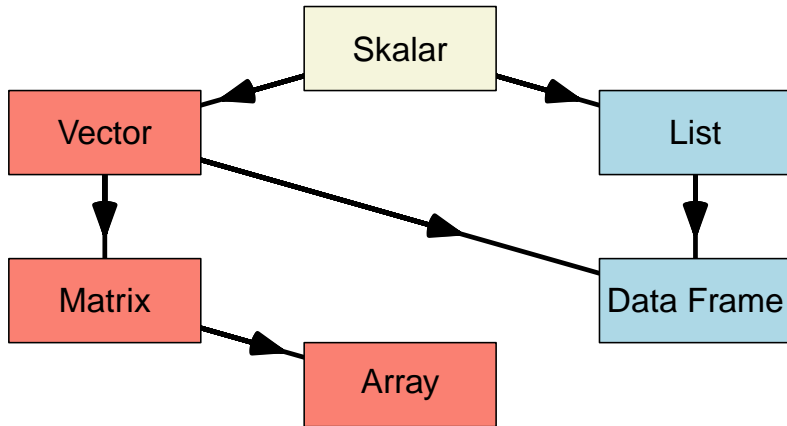
Language	local	global
R	#inside function x <- 1 x = 1	#inside function globX <-1 ;
Python	x = 1; #inside def	global globX; globX = 1; # inside def
Perl	my \$x = 1; # inside sub	my \$globX = 1; our \$GLOBX = 1; # out of sub ;

⇒ if possible avoid global variables!!!



# Data containers (data structures)

⇒ objects in which we store our variables



# Variable examples

```
#!/usr/bin/Rscript
skal_var = 3;
vect_var = c(1,2,3,4);
list_var = list('me' = 'Instructor', 'you' = 'Student!');
matr_var = matrix(1:4,nrow=2)
print(skal_var);
print(vect_var);
print(list_var[1]);
print(list_var[['me']]);
print(matr_var)
print(matr_var[1,1:2])
⇒ [1] 3
⇒ [1] 1 2 3 4
⇒ $me
⇒ [1] "Instructor"
⇒
⇒ [1] "Instructor"
```

⇒            [, 1]   [, 2]  
⇒ [1, ]       1       3  
⇒ [2, ]       2       4  
⇒ [1] 1 3



⇒ R arrays/vectors start with 1 for first element

# Data Containers

Lang	Skalar	Vector/Array	List/Hash/Dict
R	<code>x=1</code>	<code>a=c(1,2,3)</code>	<code>l=list(a=1,b=2)</code>
Perl	<code>\$x=1 ;</code>	<code>@a=(1,2,3);</code>	<code>%l=('a'=&gt;1, 'b'=&gt;2);</code>
Python	<code>x=1</code>	<code>a=[1,2,3]</code>	<code>l={'a':1, 'b':2 }</code>

# Giving out a variable

```
#!/usr/bin/Rscript
x=3;
print(x) # with automatic return and line number
cat(x,'\n') # no automatic line break and no numbering
print(paste('x is',x));
# using sprintf with formatting codes
print(sprintf('float: %.3f integer: %i ',x,x))
```

⇒ [1] 3

⇒ 3

⇒ [1] "x is 3"

⇒ [1] "float: 3.000 integer: 3 "

 varsout.r

# Getting input from the user

⇒ users should not need to write values in the source code!!

- interactive (readline)
- command line arguments (commandArgs - day 2)
- (Python:

```
> > > x = input('give a number:  '))
```

- R:

```
> x = readline(prompt='give a  
number: ')
```

- but with R readline works only in interactive mode, not in scripts :(

# Our R readline extension

```
input = function (prompt="Enter: ") {  
  if (interactive() ){  
    return(readline(prompt))  
  } else {  
    cat(prompt);  
    return(readLines("stdin",n=1))  
  }  
}  
  
x=input("Enter a numerical value: ")  
x=as.integer(x)  
print(x*12)
```

```
[groth@mandel]$ Rscript ../scripts/input.r  
Enter a numerical value: 12  
[1] 144
```

## 1.4 System Variables

```
#!/usr/bin/R --slave --no-init-file -f  
print(Sys.getenv("HOME"))  
print(Sys.getenv("USER"))  
⇒ [1] "/home/groth"  
⇒ [1] "groth"
```



On Windows:

```
> Sys.getenv("HOME")  
[1] "C:\\Users\\Dr. Detlef Groth\\Documents"  
> Sys.getenv("USER")  
[1] ""  
> Sys.getenv("USERNAME")  
[1] "Dr. Detlef Groth"
```



# 1.5 Help

## Help for R

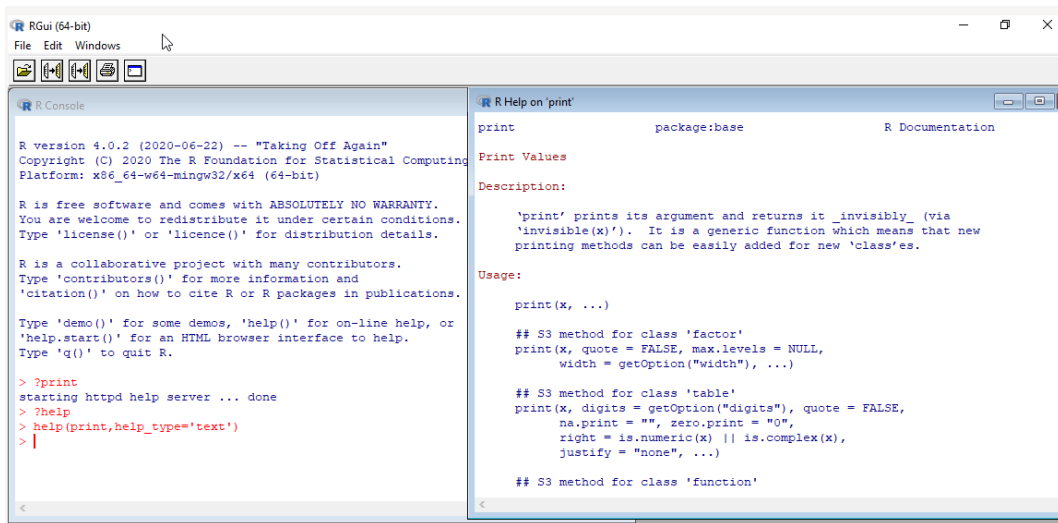
Interactive:

```
> help(print)
> ?print
> ??clust
```

Bash:

```
$ R -e 'help(print) '
$ function Rdoc { R --slave -e "help($1) " ; }
$ # this can go into .bashrc
$ Rdoc print
```

# Help on Windows



# General Help in the Terminal

- `man command`  $\Rightarrow$  top down reading
- `info command`  $\Rightarrow$  hyper text browser
- `info`  $\Rightarrow$  all info pages
- `command --help`  $\Rightarrow$  close to man pages
- `command -h`

```
$ man less
```

```
$ info less
```

```
$ man cd
```

```
$ info bash
```

```
$ less --help | less
```

# Comments: R

```
# a hash single line comment
# use it for commenting
print('Hello World!')
if (FALSE) {
  trick
  this gets never executed
  # used for larger blocks
  res=dg.longrunning.func()
}
print('Bye') # executed again
```

## 1.6 Eval

```
#!/usr/bin/R --slave --no-init-file -f  
x=3;  
eval(parse(text=paste('print(3*',x,')')));  
⇒ [1] 9
```

 eval.r

But “eval” with care, not on server for clients!!

# Summary I

- Background:
  - installs and editor
  - interactive, terminal, script file
  - editing and executingscript files
  - standalone executable files, chmod 755
- Programming:
  - variables: skalar, vector, array, list
  - user input
  - comments
  - help systems
  - system variables

# 1.7 Operators

Logical  
&, |, !, ...

Arithmetic  
+, −, \*, /...

Membership  
%in%, (%ni%)

Assignment  
=, <−, <<−, ...

Relational  
==, <=, >...

Miscellaneous  
?, %\*%, :

```
> '%ni%' = function (x,y) { !(x %in% y) }
```

# Help about Operators

- `help(TOPIC)`
- `help(Arithmetic)`
- `help(Logic)`
- `??operators`

```
> x <- 5
```

```
> x
```

```
[1] 5
```

```
> x = 3
```

```
> x
```

```
[1] 3
```

```
> x == 3
```

```
[1] TRUE
```

```
> !(x == 3)
```

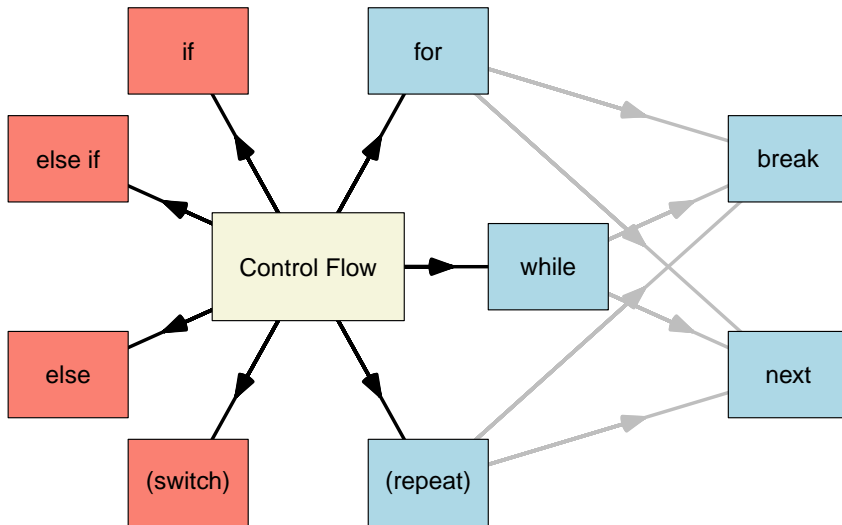
```
[1] FALSE
```

```
> c(1:6) %in% x
```

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



## 1.8 Control Flow: Conditionals



## Decision Making:

Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.

Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.

[Tutorialspoint — Python 3 – Decision Making, 2019]

- programming logic:
- `if` something is true  $\Rightarrow$  do something
- `else if` something else is true  $\Rightarrow$  do something
- `else` in any other case  $\Rightarrow$  do something
- `?Control`
- Perl and R use curly braces to structure blocks
- Python is special, it uses colon and indentation!
- van Rossum thought that programmers must be forced to properly format their code
- I personally think that this is the task of the editor, not the programmer

# Conditionals

```
#!/usr/bin/Rscript
```

```
Sys.time()
```

```
substr(Sys.time(),12,13) # get the hour part
```

```
x=as.numeric(substr(Sys.time(),12,13))
```

```
if (x > 4 && x < 11) {
```

```
  print('Good morning')
```

```
} else if (x < 14) {
```

```
  print('Good day!')
```

```
} else if (x < 18) {
```

```
  print('Good afternoon!')
```

```
} else if (x < 22) {
```

```
  print('Good evening!')
```

```
} else {
```

```
  print('Good night!')
```

```
}
```

```
⇒ [1] "2020-09-25 15:03:57 CEST"
```

```
⇒ [1] "15"
```

```
⇒ [1] "Good afternoon!"
```



# Placement of Curly Braces

```
if (cond) {          if (cond) {expr}          if (cond)
    expr              {                          {
}                                                            expr
                                                            }
```

- whitespace does not matter
- formatting just for better reading and understanding
- just a matter of personal style
- my style:
  - opening brace at the end of a line of a control flow construct (no line lost for a single brace!)
  - closing brace at the beginning of a line, or just whitespaces before
  - else if and else directly after closing brace
- your style: develop one and stick with it

## 1.9 Control Flow: Loops

### Loops:

In general, statements are executed sequentially – The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

[Tutorialspoint — Python 3 – Loops, 2019]

# Loop aim

⇒ Do operations as long as a certain condition is true (*while*) or for a certain condition or a certain number of times (*for*)!

- *for* (R, Perl, Python)
- *foreach* (Perl)
- *while* (R, Perl, Python)
- *repeat* (R)
- exit from loop:
  - *last* (Perl)
  - *break* (Python, R)
- skip rest of current and go to next iteration:
  - *next* (Perl, R)
  - *continue* (Python)

# Loops - for: R

```
#!/usr/bin/R --slave --silent --no-init-file -f
lst=list(a=1,b=2,k=3)
for (nm in names(lst)) {
  cat(paste(nm, '=', lst[[nm]], ';'))
}
cat('\n')
for (i in 1:4) {
  cat(paste(i, ' '))
}
cat('\n')
⇒ a = 1 ; b = 2 ; k = 3 ;
⇒ 1 2 3 4
```

 loopfor.R



# Loops - while: R

```
#!/usr/bin/R --slave --silent --no-init-file -f
x=1
while (x < 3) {
  x=x+1
  cat(paste('x =', x, ' ; '))
}
cat('\n')
while (TRUE) {
  if (x > 10) { break ; }
  cat(paste(x, ' '))
  x=x+1
}
cat('\n')
⇒ x = 2 ; x = 3 ;
⇒ 3 4 5 6 7 8 9 10
```

 loopwhile.R

# Summary Control Flow

- curly braces, indentation
- if, else if, else (switch)
- for, while (repeat)
- repeat == while(TRUE)
- break, next
- editor snippets

## 1.10 Functions

- **function** keyword R (def - Python, sub - Perl)
- functions are used to structure code
- organize your code in a central place
- functions can get arguments, sometimes also called *parameters*
- arguments are processed inside the function
- arguments can be optional or mandatory
- functions can return processed value(s)

## Function:

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called *user-defined functions*.

[Tutorialspoint — Python 3 – Functions, 2019]

# Preview Functions

- function keyword (Python def, Perl sub, Tcl proc, ...)
- function name
- argument definitions - input of the caller of this function
- function body - the implementation
- eventually a return statement
- more tomorrow

# Functions: R

- Intro-R: chapter 10!!
- <http://www.tutorialspoint.com/r/index.htm>

```
name <- function (arguments) {  
  function body  
  return(value)  
}
```

```
#!/usr/bin/R --slave --silent --no-init-file -f  
func.name = function (mand.arg, optional.arg=2) {  
  return(mand.arg^optional.arg);  
}  
print(func.name(2))  
print(func.name(2, 4))  
print(func.name(optional.arg=4, mand.arg=3))
```

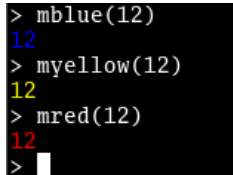
⇒ [1] 4  
⇒ [1] 16  
⇒ [1] 81



## 1.11 Colored Terminal

⇒ [https://en.wikipedia.org/wiki/ANSI\\_escape\\_code](https://en.wikipedia.org/wiki/ANSI_escape_code)

```
> mblue = function (msg) {  
+   cat(paste('\033[94m', msg, '\033[0m\n', sep=' '))  
+ }  
> mred = function (msg) {  
+   cat(paste('\033[91m', msg, '\033[0m\n', sep=' '))  
+ }  
> myellow = function (msg) {  
+   cat(paste('\033[93m', msg, '\033[0m\n', sep=' '))  
+ }
```

A terminal window with a black background. It shows the execution of three functions: mblue(12) which outputs '12' in blue, myellow(12) which outputs '12' in yellow, and mred(12) which outputs '12' in red. The prompt character is a white square.

```
> mblue(12)  
12  
> myellow(12)  
12  
> mred(12)  
12  
> 
```



	90
31	91
32	92
33	93
34	94
35	95
36	96
37	97

⇒ see also R-library crayon

# Colored Terminal with global variables

```
GREY=GRAY='\033[90m'  
RED='\033[91m'  
GREEN='\033[92m'  
YELLOW='\033[93m'  
BLUE='\033[94m'  
MAGENTA='\033[95m'  
CYAN='\033[96m'  
WHITE='\033[97m'  
RESET='\033[0m'  
cat(paste(RED, 'Hello Red World!\n', RESET))  
cat(paste(GREEN, 'Hello Green World!\n', RESET))  
cat('Hello Normal World!\n')
```

```
> GREY=GRAY='\033[90m'  
> RED='\033[91m'  
> GREEN='\033[92m'  
> YELLOW='\033[93m'  
> BLUE='\033[94m'  
> MAGENTA='\033[95m'  
> CYAN='\033[96m'  
> WHITE=BLUE='\033[97m'  
> RESET=BLUE='\033[0m'  
> cat(paste(RED, 'Hello Red World!\n', RESET))  
Hello Red World!  
> cat(paste(GREEN, 'Hello Green World!\n', RESET))  
Hello Green World!  
> cat('Hello Normal World!\n')  
Hello Normal World!
```

```
#!/usr/bin/Rscript
showAnsi = function() {
  for (i in 0:10) {
    for (j in 0:9) {
      n = 10*i + j;
      if (n > 109) break;
      cat(sprintf("\033[%dm %3d\033[m", n, n))
    }
    cat("\n");
  }
  return (0);
}
showAnsi()
```

```
[groth@bariuke build]$ ./ansi.R
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109
[1] 0
```

# Hints

- avoid global variables if possible
- personal coding styles for indentation
- DG:
  - indent with four spaces
  - opening brace not on own line
  - only final closing brace on own line
- configure Geany for a template (snippet)!!
- (for applications always have a function *main* which should start your program, place it at the end)
- (before *main* other function definitions)
- (check if the script is interpreted directly, if yes execute *main*)

# Summary II

- variables: local vs global
- data types integer, float, string, boolean, ...
- data structures scalar, vector, array, list
- operators: math, logical
- conditionals: if, else if, else
- loops: for, while and break and next
- ANSI codes for colored terminal
- tomorrow:
  - functions
  - command line arguments
  - file input and output

# 1.12 Exercise Introduction

## Aim:

- install the required tools, editor, R
- prepare a directory for todays session
- make you comfortable with the editor
- write simple hello world scripts
- control flow exercises
- syntax highlighting
- code snippets

# Task 1 - Installation Geany

- install the Geany text editor
- Linux, Unix use the package manager
- Fedora: `sudo dnf install geany`
- Debian/Ubuntu: `sudo apt-get install geany`
- Windows/MacOSX - manual download and install from:  
<https://www.geany.org/download/releases/>
- after install start the Geany text editor (geany)

## Task 2 - Install R

- install the R programming tools
- Linux and other Unixes (MacOSX(?), BSD, ...) use the package manager
- Fedora: `sudo dnf install R`
- Debian/Ubuntu: `sudo apt-get install R`
- Windows/MacOSX - manual download and install from:  
<https://lib.ugent.be/CRAN/>
- after install start R (R)
- do a simple calculation `> 1+1` and then `> 1+2==2` (two equal signs!)
- check out what is your home directory by writing  
`Sys.getenv("HOME")` (UNIX) or  
`Sys.getenv("USERPROFILE")` (Windows)  
in the R terminal after



## Task 3 - Creating a “Hello World” script

- get the geany editor running: `$ geany &`
- open the terminal inside geany (Linux)
- create a directory inside your home called `PwR-labs`
- create the a R hello world scripts from the lecture at slide 22 (`hw.R`)
- store the script files inside the new `PwR-labs` directory
- UNIX (Linux, OSX):
  - use `chmod 755` to make them running standalone
  - run them in your terminal, check output
- Configure the geany text editor, that it can run R scripts directly by using the Geany menu point 'build->set build command'
- below at “execute commands” write the command line to execute R scripts

- Linux: `R --vanilla --slave %f`
- Windows: `"C:\Program Files\R\R-4.0.2\bin\x64\Rterm.exe" --vanilla --slave "%f"`
- MacOSX: `R --vanilla --slave "%f"`

## Task 4 - Extending “Hello World!”

- extend the hello world script, so that it prints as well the home directory name `studNNNN`
- extend the script that hello world is printed in green color on the terminal by using a variable `green` as can be seen in the lecture
- extend the script, that it displays as well the user name in the terminal in red color `Sys.getenv('USERNAME')` (Windows), `Sys.getenv('USER')` (Unix)
- save those scripts with a new name like `hw-user.pl`
- make them running standalone using `chmod 755 filename`
- replace `filename` with the name of the script

## Task 5 - Dealing with input

- create multiplier scripts for R: `mymulti.R`
- you should place the input function (see below) at the beginning of your script
- the user should be asked for two numbers and then the scripts multiplies both numbers and prints the result
- please note, that you have to convert the input to a number by using the R function `as.numeric`
- create an `exercise.R` script which creates two random numbers using the R `sample` function
- `sample(1:20,2)` samples two numbers in the range from 1 to 20
- write a multiplication questions with those two random numbers to the user
- compare the given user answer with the real solution
- write wrong or false in red and green to the terminal depending on the correctness of the answer
- use a for loop to ask 5 questions one after the other

```
input = function (prompt="Enter: ") {  
  if (interactive() ){  
    return(readline(prompt))  
  } else {  
    cat(prompt);  
    return(readLines("stdin",n=1))  
  }  
}  
# comment for usage example  
# x=input("Enter a numerical value: ")  
# x=as.integer(x)
```

⇒ finish the task until tomorrow

⇒ have a look at: [https://www.tutorialspoint.com/r/](https://www.tutorialspoint.com/r/chapters)  
chapters until decision making and loops

# Programming with R

## Day 2 - Functions

### University of Potsdam

Detlef Groth  
2020

# Last Lecture

- install R
- install Geany
- variables
- global, local scope (functions)
- data types
- data structures
- help
- ANSI colors

# 2 Functions

## Outline

2.1	Definition and code organization . . . . .	97
2.1.1	Definition . . . . .	97
2.1.2	The main function . . . . .	102
2.1.3	Function arguments . . . . .	106
2.2	File IO . . . . .	114
2.3	Command line arguments . . . . .	126
2.4	Terminal applications . . . . .	139
2.5	Exercise Functions . . . . .	154

# Outline

## Day 1 (basics)

- setup, install  
editor, simple programs
- variables, operators
- data structures, control flow

## Day 2 (basics)

- **functions**
- **file input/output**
- **terminal interaction**
- **command line arguments**

## Day 3 (advanced)

- object oriented  
programming
- code documentation
- R base graphics system

## Day 4 (advanced)

- using packages
- writing packages
- package documentation

## Day 5 (advanced)

- graphical user interfaces
- tcltk (shiny)



## 2.1 Definition and code organization

### 2.1.1 Definition

- **function** keyword in R
- functions are used to structure code
- organize your code in a central place
- goal: code reuse
- functions can get *arguments*, sometimes also called *parameters*
- arguments are processed inside the function
- arguments can be mandatory or optional
- R supports named arguments, flexible order possible
- functions can *return* processed value(s)

## Function:

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called *user-defined functions*.

[Tutorialspoint — Python 3 – Functions, 2019]

# Functions

- keyword (R function, Python def, Perl sub, ...)
- function name
- argument definition(s) - input by caller
- function body - the implementation
- eventually a return statement

# Functions: R

- Intro-R: chapter 10!!
- <http://www.tutorialspoint.com/r/index.htm>

```
name <- function (argument(s)) {  
  function body  
  return(value)  
}  
  
#!/usr/bin/env Rscript  
func.name = function (mand.arg,optional.arg=2) {  
  return(mand.arg^optional.arg);  
}  
print(func.name(2))  
print(func.name(2,2))  
print(func.name(2,4))  
print(func.name(optional.arg=4,mand.arg=3))
```

⇒ [1] 4  
⇒ [1] 4  
⇒ [1] 16  
⇒ [1] 81



## 2.1.2 The main function

In applications, not libraries, you should have a function `main` at the end of your code which is the entry point of your programming logic. You can further check if the R-script file was executed directly. So, if called with `R --vanilla -f filename.R` or using `Rscript filename.R` `main` will be automatically executed. Languages like C and C++ will automatically call a main function. In R, Python and many other programming languages, this is not required but considered good style and practice.

```
# filename.R
func1 <- function(x,y) {
  # ...
  func2(x+y)
}

func2 <- function(x) { # ... }

main <- function () {
  func1(1,2)
  # ...
}

# check if this is the script sourced
# first in the interpreter
if (sys.nframe() == 0L & !interactive()) {
  main()
}
```

## Simple example: main.R

```
#!/usr/bin/Rscript
main <- function () {
  cat("Hello Main!\n");
  # ...
}
# check if this is the main script
if (sys.nframe() == 0L & !interactive()) {
  main()
}
```

```
[groth@bariuke build]$ chmod 755 ../main.R
[groth@bariuke build]$ ../main.R
Hello Main!
```



```
[groth@bariuke build]$ R --vanilla
```

```
R version 3.6.3 (2020-02-29) -- "Holding the  
Copyright (C) 2020 The R Foundation for Stat  
Platform: x86_64-redhat-linux-gnu (64-bit)
```

```
...
```

```
Type 'q()' to quit R.
```

```
> source("../main.R") # no output below  
>
```

⇒ The file `main.R` can be sourced within an interactive session without calling the main function.

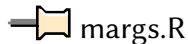
## 2.1.3 Function arguments

- mandatory (required, no default)
- optional (with defaults)
- three dots (takes and delegates any additional argument)

# Mandatory arguments

- no default value given in argument definition
- can't be omitted if calling the function

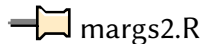
```
#!/usr/bin/env Rscript
incr <- function (x,y) {
  print(paste('x is:',x))
  x=x+y
  return(x)
}
print(incr(2,5))
print(incr(y=5,x=2)) # named: order switch possible
⇒ [1] "x is: 2"
⇒ [1] 7
⇒ [1] "x is: 2"
⇒ [1] 7
```



# Optional arguments

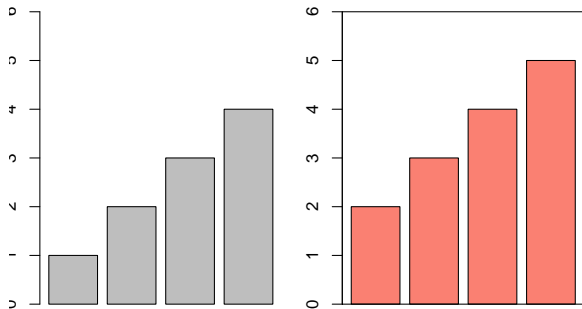
- optional arguments have a useful default
- can be omitted, then default is used

```
#!/usr/bin/env Rscript
mround <- function (x,digit=3) {
  return(round(x,digit))
}
print(mround(2.12345))
print(mround(2.12345,2))
print(mround(2.12345,digit=2)) # preferred way
print(mround(digit=2,x=2.12345))
⇒ [1] 2.123
⇒ [1] 2.12
⇒ [1] 2.12
⇒ [1] 2.12
```




# The delegation argument: ...

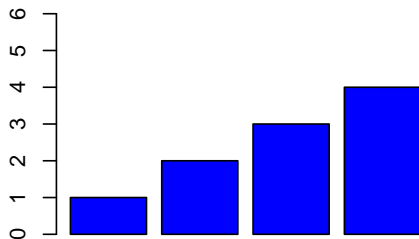
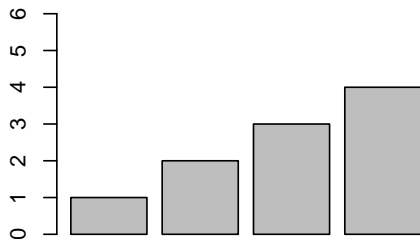
```
> par(mfrow=c(1,2),mai=c(0.3,0.3,0.3,0.3))
> mbarplot = function(data,col='salmon',...) {
+   barplot(data,col=col,...)
+   box()
+ }
> barplot(1:4,
+   ylim=c(0,6))
> mbarplot(2:5,
+   ylim=c(0,6))
```



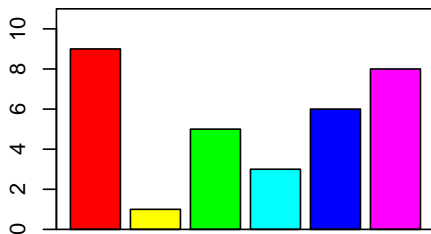
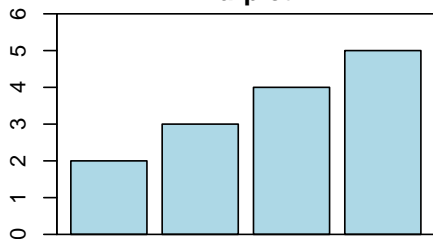
# Same as PDF

```
#!/usr/bin/env Rscript
pdf('barplot.pdf',width=6,height=4)
par(mfrow=c(2,2),mai=c(0.3,0.3,0.3,0.3))
mbarplot = function (data,col='salmon',...) {
  barplot(data,col=col,...)
  box()
}
barplot(1:4,ylim=c(0,6))
barplot(1:4,ylim=c(0,6),col='blue')
mbarplot(2:5,ylim=c(0,6),col='light blue',
  main='Barplot')
mbarplot(sample(1:10,6), col=rainbow(6),ylim=c(0,11))
dev.off()
⇒ null device
⇒ 1
```

 barplot.R



**Barplot**




# Summary demonstration

```
#!/usr/bin/env Rscript
f1 <- function (z) {
  print('inside f1')
  print(z^2)
}
main <- function (x, y=2, ...) {
  print('inside main')
  print(x^2+y)
  if (!missing(...)) {
    f1(...)
  }
}
if (sys.nframe() == 0L & !interactive()) {
  main(2)
  main(2,10)
  main(2,10,4)
  main(z=4,y=10,x=2)
}
```



```
⇒ [1] "inside main"  
⇒ [1] 6  
⇒ [1] "inside main"  
⇒ [1] 14  
⇒ [1] "inside main"  
⇒ [1] 14  
⇒ [1] "inside f1"  
⇒ [1] 16  
⇒ [1] "inside main"  
⇒ [1] 14  
⇒ [1] "inside f1"  
⇒ [1] 16
```

 args.R

⇒ prepare your Geany snippets!

## 2.2 File IO

- input
  - files
  - stdin
- output:
  - files
  - stdout
  - stderr

# read.table

⇒ Standard reading of csv files.

- read.table
- read.csv
- read.csv2
- read.<TAB> ...

# GFF files

```
> sdata=read.table(  
+   ".../.../data/uniprot-proteome-UP000000354.gff",  
+   sep="\t")  
> dim(sdata)  
[1] 292  10  
> sdata[1:4,1:5]  
      V1      V2      V3 V4 V5  
1 P59637 UniProtKB      Chain  1 76  
2 P59637 UniProtKB Topological domain  1 16  
3 P59637 UniProtKB      Transmembrane 17 37  
4 P59637 UniProtKB Topological domain 38 76  
> table(sdata[, 'V1'], sdata[, 'V3'])[1:4, 1:3]  
      Beta strand Chain Coiled coil  
P0C6X7      0      15      0  
P59594     74      0      2  
P59595     12      1      0  
P59596      0      1      0
```

## `openxlsx::read.xlsx`

- Can be used to read and write Excel xlsx files.
- Xlsx files are standardized spreadsheet files.
- There are many packages which can read Excel files.
- `openxlsx` is my favourite as it is very fast, nicely documented and can be used even to embed plots into output Excel files.

⇒ `read.xlsx` and `read.table` are great for tabular data which fit into memory, but what about files which are huge and/or which are not in tabular format?

# Non-tabular data: How to copy a file?

⇒ let's ignore that we have `file.copy` in R

```
#!/usr/bin/env Rscript
fin  = file('../main.R', "r")
fout = file('main-copy.R', "w")
while (length((line = readLines(fin,n=1L)))>0) {
  cat(paste(line, "\n", sep=""), file=fout)
}
close(fin)
close(fout)
print('Copy of file made!')
⇒ [1] "Copy of file made!"
```

➡ fcopy.R

```
#!/bin/sh
ls -l main-copy.R ../main.R
⇒ -rwxr-xr-x. 1 groth groth 174 Sep 25 15:04 main-copy.R
⇒ -rwxr-xr-x. 1 groth groth 174 Sep 11 13:56 ../main.R
```

➡ diff.sh

# Geany snippets

```
fin=fin = file('%cursor%', 'r')\nwhile(length((  
    line=readLines(fin,n=1L)))>0) {\n    print(line)\n}\nnclose(fin)  
fout=fout=file('%cursor%', 'w')\nncat('Hello\n',  
    file=fout)\nnclose(fout)\n
```

⇒ remove returns on first definition line

# Example input

```
#!/bin/sh
```

```
head -n 5 ../../data/uniprot-proteome-UP000000354.* | cut
```

```
⇒ ==> ../../data/uniprot-proteome-UP000000354.fasta
```

```
⇒ >sp|P59637|VEMP_CVHSA Envelope small membrane prot
```

```
⇒ MYSFVSEETGTLIVNSVLLFLAFVVFLVTLAILTALRLCAYCCNIVNVS
```

```
⇒ RVKNLNSSEGVDPDLLV
```

```
⇒ >sp|P59596|VME1_CVHSA Membrane protein OS=Human SA
```

```
⇒ MADNGTITVEELKQLLEQWNLVIGFLFLAWIMLLQFAYSNRNRFlyIIKL
```

```
⇒
```

```
⇒ ==> ../../data/uniprot-proteome-UP000000354.gff <=
```

```
⇒ ##gff-version 3
```

```
⇒ ##sequence-region P59637 1 76
```

```
⇒ P59637 UniProtKB Chain 1 76
```

```
⇒ P59637 UniProtKB Topological domain
```

```
⇒ P59637 UniProtKB Transmembrane 17
```

 fasta.sh



# R: file.show

```
#!/usr/bin/env Rscript
file.show(' ../ ../data/uniprot-proteome-UP000000354.fasta'
, pager='head')

⇒ >sp|P59637|VEMP_CVHSA Envelope small membrane protein O
⇒ MYSFVSEETGTLIVNSVLLFLAFVVFLLVTLAILTALRLCAYCCNIVNVSLVKPT
⇒ RVKNLNSSEGVPDLLV
⇒ >sp|P59596|VME1_CVHSA Membrane protein OS=Human SARS co
⇒ MADNGTITVEELKQLLEQWNLVIGFLFLAWIMLLQFAYSNRNRFlyIIKLVFLWL
⇒ LACFVLAAYRINWVTGGIAIAMACIVGLMWLSYFVASFRLFARTRSMWSFNPET
⇒ VPLRGTIVTRPLMESELVIGAVIIRGHLMAGHSLGRCDIKDLPKEITVATSRTL
⇒ GASQRVGTDSGFAAYNRYRIGNYKLNTDHAGSNDNIALLVQ
⇒ >sp|Q7TLC7|Y14_CVHSA Uncharacterized protein 14 OS=Huma
⇒ MLPPCYNFLKEQHCQKASTQREAEAAVKPLLAPHHVVAVIQEIQLLAAVGEILL
```

 fasta.R

But this does not work on windows!

# file and dir commands

```
> file.<TAB>
```

```
file.access    file.copy      file.exists    file.mode  
file.remove    file.size      file.append    file.create  
file.info      file.mtime     file.rename    file.symlink  
file.choose    file.edit      file.link      file.path  
file.show
```

```
> dir.<TAB>
```

```
dir.create     dir.exists
```

⇒ Exercise: Write a `file.head` function which displays the first 6 lines of a file regardless of the pager! So, it should work as well on Windows.

# file.head

```
#!/usr/bin/env Rscript
file.head = function (filename,n=6) {
  if (!file.exists(filename)) {
    stop(paste('Error! File',filename,'does not exist!'))
  }
  fin=file(filename, 'r')
  i=0
  while(length((line=readLines(fin,n=1L)))>0) {
    i = i +1
    cat(line, '\n')
    if (i == n) {
      break
    }
  }
  close(fin)
}
file.head('../ ../data/uniprot-proteome-UP000000354.fasta'
, n=3)
```

⇒ >sp|P59637|VEMP\_CVHSA Envelope small membrane protein O  
⇒ MYSFVSEETGTLIVNSVLLFLAFVVFLLVTLAILTALRLCAYCCNIVNVSLVKPT  
⇒ RVKNLNSSEGVPDLLV

 fhead.R

# seqinr::read.fasta

## read.fasta

From [seqinr v3.6-1](#)  
by [Simon Penel](#)

99.99th  
Percentile

### Read FASTA Formatted Files

Read nucleic or amino-acid sequences from a file in FASTA format.

### Usage

```
read.fasta(file = system.file("sequences/ct.fasta.gz", package = "seqinr"),  
  seqtype = c("DNA", "AA"), as.string = FALSE, forcedNATolower = TRUE,  
  set.attributes = TRUE, legacy.mode = TRUE, seqonly = FALSE, strip.desc = FALSE,  
  whole.header = FALSE,  
  bfa = FALSE, sizeof.longlong = .Machine$sizeof.longlong,  
  endian = .Platform$endian, apply.mask = TRUE)
```

⇒ In the course we will write our own FASTA package

## 2.3 Command line arguments

- As you submit arguments to functions you can submit arguments to your application on the terminal.
- Example: `Rscript app.R arg1 arg2 arg3`
- Goal: Set variables in your application at runtime, not by writing the values into the source code.
- Example:  
`firefox https://www.uni-potsdam.de`
- Firefox has not hardcoded the URL into its source code, but you can still load the Uni web page

# C and command line arguments

It is possible to pass some values from the command line to your C programs when they are executed. These values are called command line arguments and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using `main()` function arguments where `argc` refers to the number of arguments passed, and `argv[]` is a pointer array which points to each argument passed to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly.

[https://www.tutorialspoint.com/cprogramming/c\\_command\\_line\\_arguments.htm](https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm)

# R: commandArgs - function

```
> ?commandArgs
```

```
commandArgs
```

```
package:base
```

```
R Documentation
```

```
Extract Command Line Arguments
```

```
Description:
```

```
Provides access to a copy of the command line  
arguments supplied when this R session was invoked.
```

```
Usage:
```

```
commandArgs(trailingOnly = FALSE)
```

```
Arguments:
```

```
trailingOnly: logical. Should only arguments after  
?--args? be returned?
```



# commandArgs example

```
#!/usr/bin/env Rscript
options(width=55)
print('All:')
print(commandArgs())
print('Trailing Only:')
print(commandArgs(trailingOnly=TRUE))
⇒ [1] "All:"
⇒ [1] "/usr/lib64/R/bin/exec/R" "--slave"
⇒ [3] "--no-restore"                "--file=./commandArgs.R"
⇒ [1] "Trailing Only:"
⇒ character(0)
```

📄 commandArgs.R

```
#!/usr/bin/sh
./commandArgs.R some arg --arg1 value1--args2 val2
⇒ [1] "All:"
⇒ [1] "/usr/lib64/R/bin/exec/R"
⇒ [2] "--slave"
```

```
⇒ [3] "--no-restore"  
⇒ [4] "--file=./commandArgs.R"  
⇒ [5] "--args"  
⇒ [6] "some"  
⇒ [7] "arg"  
⇒ [8] "--arg1"  
⇒ [9] "value1--args2"  
⇒ [10] "val2"  
⇒ [1] "Trailing Only:"  
⇒ [1] "some" "arg" "--arg1"  
⇒ [4] "value1--args2" "val2"
```


 commandArgs.sh

⇒ Access to arguments by index!


# More complex example

```
#!/usr/bin/env Rscript
options(width=55)
getArgs = function () {
  l=list()
  args=commandArgs(trailingOnly=TRUE)
  i = 0
  for (arg in args) {
    i = i +1
    if (grepl('^--',arg)) {
      l[[arg]]=args[i+1]
      next
    }
  }
  return(l)
}
args=getArgs()
print(args)
print(args[['--infile']])
```

```
⇒ list()  
⇒ NULL
```

 commandArgs2.R

```
#!/usr/bin/sh  
./commandArgs2.R --infile test.fasta --outfile out.txt \  
    --arg1 value1  
⇒ $`--infile`  
⇒ [1] "test.fasta"  
⇒  
⇒ $`--outfile`  
⇒ [1] "out.txt"  
⇒  
⇒ $`--arg1`  
⇒ [1] "value1"  
⇒  
⇒ [1] "test.fasta"
```

 commandArgs2.sh

# R library argparse (needs Python!!)

argparse: Command Line Optional and Positional Argument Parser

A command line parser to be used with Rscript to write "#!" shebang scripts that gracefully accept positional and optional arguments and automatically generate usage.

Version: 2.0.1  
Imports: [R6](#), [findpython](#), [jsonlite](#)  
Suggests: [covr](#), [knitr](#) (≥ 1.15.19), [testthat](#)  
Published: 2019-03-08  
Author: Trevor L Davis [aut, cre], Allen Day [ctb] (Some documentation and examples ported from the getopt package.), Python Software Foundation [ctb] (Some documentation from the optparse Python module.), Paul Newell [ctb]  
Maintainer: Trevor L Davis <trevor.l.davis at gmail.com>  
BugReports: <https://github.com/trevorld/r-argparse/issues>  
License: [GPL-2](#) | [GPL-3](#) [expanded from: GPL (≥ 2)]

⇒ this R-package has dubious dependencies!

⇒ integrating several programming languages is fun first but ends with pain later ...

# R library argparser (no dependencies!!)

argparser: Command-Line Argument Parser

Cross-platform command-line argument parser written purely in R with no external dependencies. It is useful with the Rscript front-end and facilitates turning an R script into an executable script.

Version: 0.6  
Depends: methods  
Published: 2019-12-17  
Author: David J. H. Shih  
Maintainer: David J. H. Shih <djh.shih at gmail.com>  
BugReports: <https://bitbucket.org/djhshih/argparser/issues>  
License: [GPL \(≥ 3\)](#)  
URL: <https://bitbucket.org/djhshih/argparser>  
NeedsCompilation: no

⇒ KISS: avoid packages with many dependencies if you can!  
⇒ No installation waiting on users computer if you install your script

# Argparser example:

```
#!/usr/bin/env Rscript
# install.packages('argparser')
library(argparser)
p <- arg_parser("A text file modifying program")
# Add a positional argument
p <- add_argument(p, "input", help="input file")
# Add an optional argument
p <- add_argument(p, "--output", help="output file", default="")
# Add a flag
p <- add_argument(p, "--append", help="append to file", flag_only = TRUE)
# Add multiple arguments together
p <- add_argument(p,
  c("ref", "--date", "--sort"),
  help = c("reference file", "date stamp to use", "sort flag"),
  flag = c(FALSE, FALSE, TRUE))
print(p) # goes to stderr :(
print('done')
```

⇒ [1] "done"

 argparser.R

```
[groth@bariuke build]$ ./argsparser.R
usage: ./argsparser.R [--] [--help] [--append]
      [--sort] [--opts OPTS] [--output OUTPUT]
      [--date DATE] input ref
```

A text file modifying program

positional arguments:

input	input file
ref	reference file

flags:

-h, --help	show this help message and exit
-a, --append	append to file
-s, --sort	sort lines



optional arguments:

-x, --opts OPTS	RDS file containing argument values
-o, --output OUTPUT	output file [default: output.txt]
-d, --date DATE	date stamp to use

- ⇒ The argparse should support all things you ever need.
- ⇒ Only GPL might be an issue if you work in a company.
- ⇒ I have my own small object oriented implementation, MIT licensed (argvparse.R).

# Summary

Standard R:

- `commandArgs`
- parse by position/index
- `trailingOnly` option

Library `argparser`:

- if you need more
- double dash and short option support
- `arg_parser`
- `add_argument`
- prints on `stderr` (I send an issue request for this)

⇒ We focus here on `commandArgs` and eventually my `getArgs` function shown above.

## 2.4 Terminal applications

### User interfaces

Command line interfaces (CLI):

- simple to implement but powerful
- sometimes tricky for ordinary users to use
- easy pipelining of applications
- good for automation

Graphical user interface (GUI):

- more difficult to implement
- easier to use for ordinary users
- difficult to automate and to connect to other tools

# Guidelines

- provide a help message if not the right arguments are given
- help message should give:
  - application name
  - short description what the program does
  - available options with understandable messages
  - (version number)
  - (author name and company)
  - (license, PD, GPL or MIT)

⇒ Even if you are the only user of the program (first three points)!

⇒ You should not need to read the source code to know what the program is doing

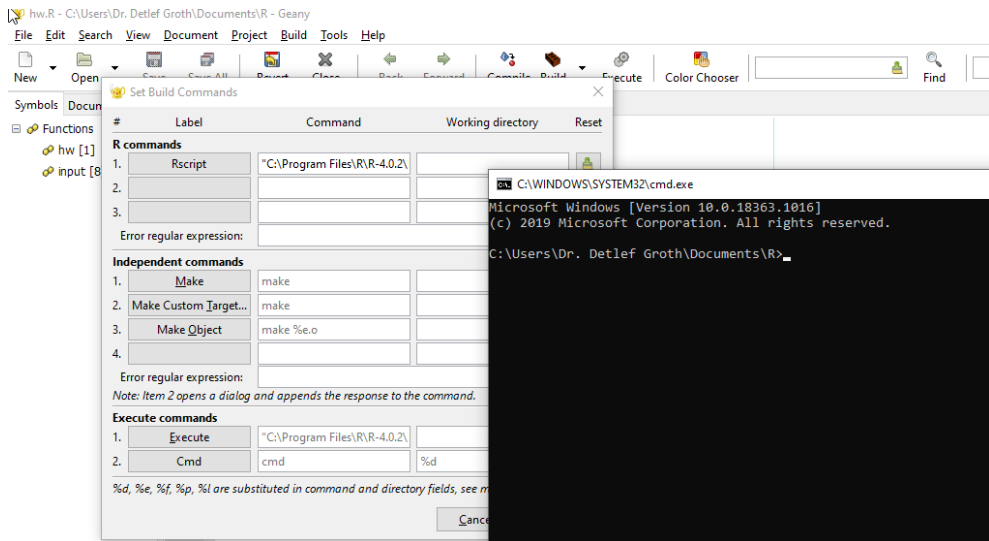
# General application outline

Homework: Create a snippet app which outlines a standard application with the following code:

```
#!/usr/bin/env Rscript
usage <- function () {
  cat("Application name by Author, Uni Potsdam, 20
  cat("Usage: appname --arg value\n")
}
main <- function (args) {

}
if (sys.nframe() == 0L & !interactive()) {
  main(commandArgs(trailingOnly=TRUE))
}
```

# Geany and cmd - Terminal




# Terminal Menu

⇒ How to write a menu for the terminal: while (TRUE)

```
#!/usr/bin/env Rscript
source('../scripts/terminal.R')
menu = function () {
  while (TRUE) {
    number=input('guess a number beween 1 and 3
      (q to quit): ')
    if (number == 'q') {
      break
    }
    if (as.integer(number) == sample(1:3,1)) {
      cat(GREEN, 'You guessed correct!\n',RESET);
    } else {
      cat(RED, 'You guessed wrong!\n',RESET);
    }
  }
}
```

```
#menu()  
print('done');  
⇒ [1] "done"
```

 guess.R

```
[groth@bariuke build]$ ./guess.R  
guess a number between 1 and 3 (q to quit): 1  
You guessed correct!  
guess a number between 1 and 3 (q to quit): 2  
You guessed correct!  
guess a number between 1 and 3 (q to quit): 2  
You guessed wrong!  
guess a number between 1 and 3 (q to quit): 1  
You guessed wrong!  
guess a number between 1 and 3 (q to quit): 3  
You guessed wrong!  
guess a number between 1 and 3 (q to quit): 1  
You guessed wrong!  
guess a number between 1 and 3 (q to quit): 2  
You guessed correct!  
guess a number between 1 and 3 (q to quit): q  
[groth@bariuke build]$
```



# terminal.R

```
input = function (prompt="Enter: ") {  
  if (interactive() ){  
    return(readline(prompt))  
  } else {  
    cat(prompt);  
    return(readLines("stdin",n=1))  
  }  
}  
  
GREY=GRAY='\033[90m'  
RED='\033[91m'  
GREEN='\033[92m'  
YELLOW='\033[93m'  
BLUE='\033[94m'  
MAGENTA='\033[95m'  
CYAN='\033[96m'  
WHITE='\033[97m'
```

RESET=' \033[0m '

# Splitting code in files

How to organize applications where the code is splitted into different files?

We would like to reuse functions and may be even use global variables over several files.

Solutions:

- `library`: assemble several files into a library (Day 4)
- `source`: load individual R source code files
- Problem with `source` is: Where is the file to find in dependence from the working directory?
- I would like to be able to source files in a directive relative to the current file.
- There is no inbuild solution, but a few workarounds.

# The problem

```
#!/usr/bin/env Rscript  
message("Hello")  
source("other.R")
```

⇒ This does only work if we are in the directory where  
other.R is.

# Workaround where is the currently sourced file?

```
thisFile <- function() {  
  cmdArgs <- commandArgs(trailingOnly = FALSE)  
  needle <- "--file="  
  match <- grep(needle, cmdArgs)  
  if (length(match) > 0) {  
    # Rscript  
    return(normalizePath(sub(needle, "",  
                           cmdArgs[match])))  
  } else {  
    # 'source'd via R console  
    return(normalizePath(sys.frames()[[1]]$ofile  
  )  
}
```

# Alternative

- Create one single big R-file
- concat all your R files into a single R file to make a running application
- ...

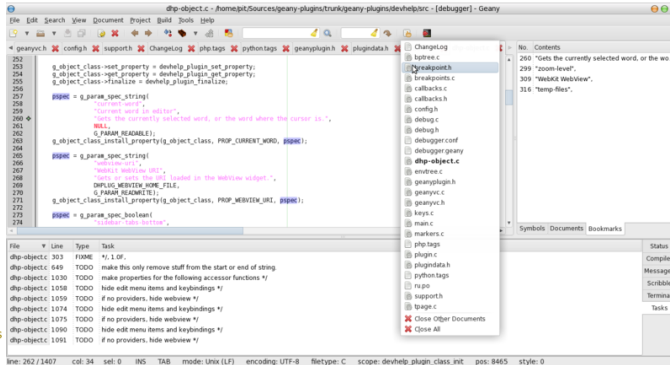
# Geany Plugins



## Plugins for Geany

### Geany Plugins project

Addons  
Autoclose  
Automark  
Codenav  
Commander  
Debugger  
Defineformat  
Devhelp  
Geanyctags  
Geanydoc  
Geanyextrasel  
Geanygendoc  
Geanyinsertnum  
Geanylua  
Geanymacro  
Geanyminiscript  
Geanynumberedbookmarks  
Geanypp  
Geanyprj  
Geanypy  
Geanyvc  
Geniuspaste  
Git-Changebar



### Contents

- About
- Features

# Plugins Install

- Fedora: `sudo dnf install geany-plugins-*`
- CentOS: `yum install geany-plugins-*`
- Ubuntu: `sudo apt-get install geany-plugins`
- Windows: no package manager, use the download page
- Mac OSX I don't know, build them yourself?



# Geany Plugins Download

https://plugins.geany.org/downloads.html



me on GitHub



## Plugins for Geany

### Geany Plugins project

- Addons
- Autoclose
- Automark
- Codenav
- Commander
- Debugger
- Defineformat
- Devhelp
- Geanyctags
- Geanydoc
- Geanyextrasel
- Geanygendoc
- Geanyinsertnum

### Download Geany-Plugins

The latest version is 1.36, you should take a look at the [release notes](#).

Download	Type	SIG	MD5 / SHA1
<a href="#">geany-plugins-1.36.tar.gz</a>	Source Code	<a href="#">GPG Signature</a>	76501a5adb92633cc41d0b6453692454
<a href="#">geany-plugins-1.36.tar.bz2</a>	Source Code	<a href="#">GPG Signature</a>	91fb4634953702f914d9105da7048a33
<a href="#">geany-plugins-1.36_setup.exe</a>	Win32 Installer	<a href="#">GPG Signature</a>	a358595dae47db32ee6ceb206bdb3dbc

[Old versions of Geany-Plugins can also be downloaded.](#)

## 2.5 Exercise Functions

### Task 0 - Workspace preparation

The goal of todays exercise today is to write a R-script which creates a tabular output file with the two columns sequence-id and sequence-length from a given FASTA file.

Steps:

- start the Geany text editor
- the files from last days exercise should be visible again
- activate the split window plugin (Tools-Plugin Manager) and assign the keys horizontal split (Ctrl-F2), vertical split (Ctrl-F3) and unsplit (Ctrl-F1)
- Windows: prepare the Cmd tool as described in the lecture for opening a terminal directly from geany in your current folder
- Windows: create a *Rscript.bat* file in this folder to execute directly your commands in this folder:

```
"C:\Program  
Files\R\R-4.0.2\bin\x64\Rscript.exe" %1 %2 %3  
%4 %5 %6 %7
```

The percent signs are place holders for possible command line arguments

- download the Fasta files from Moodle and save them in a directory data below of your R script
- look into the FASTA files using geany and rename them to sars-cov1.fasta and sars-cov2.fasta depending on the content
- so your directory stucture should look like this:

```
data  
data/sars-cov1.fasta  
data/sars-cov2.fasta  
hw.R  
Rscript.bat (if on Windows)
```

Windows: Try to execute “Rscript.bat hw.” in the Cmd-Terminal

# Task 1 - Write a BMI function

- lets start with two simple functions
- R has the function `mean` to calculate the mean from a numeric vector.
- see below the mean for the numbers 1:10
- thereafter an implementation of our own mean function: `mymean`

```
> mean(1:10)
[1] 5.5
> mymean <- function (x) { return(sum(x)/length(x)) }
> mymean(1:10)
[1] 5.5
```

Task 1: The body mass index (BMI) is calculated as  $BMI = \frac{Kg}{m^2}$ . Write your own BMI function and compare your results with this online tool: [https://www.nhlbi.nih.gov/health/educational/lose\\_wt/BMI/bmi-m.htm](https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmi-m.htm).

## Task 2 - Outline application

### Count number of sequences in FASTA file

Hint: if not done yet create the snippets `fin` and `fout` for your text editor as shown in the lecture.

Functions to use:

- `commandArgs`
- `file.exists` (check if a file exists)

Workflow:

- create a new empty file and save it as `fastainfo.R`
- add the shebang line `'#!/usr/bin/env Rscript'` at top of your file even if you are on Windows
- your script might then Run as well on Windows
- add a main and a help function (later to a Geany snippet if not done yet)
- use the `commandArgs` function with `trailingOnly=TRUE` to parse command line arguments

- if there is no command line argument (or there is no file with the given name) call the help function

## Outline:

shebang

help-function

sys.nframe check

main-function with

\* commandArgs

\* file.exists

## Task 3 - Implementation: Count sequence lengths in FASTA file

Functions to use:

- file (open a file in read or write mode)
- grepl (check if a text pattern exists in a string, l stands for logical)
- gsub (replacement of text)
- nchar (number of characters)
- to your application add countFasta function
- arguments to this function should be a filename
- again do a check for the file existence (Why again - who knows?)
- open and loop over the file
- if a line starts with a greater sign, checked with grepl we have a header
- we then extract the id using gsub

```
> grepl('^>', 'MACTR')  
[1] FALSE
```

```
> grepl('^>', '>ID01 some more text')  
[1] TRUE  
> gsub('^>([ ^ ]+).+', '\\1', '>ID01 some more text')  
[1] "ID01"
```

Those are regular expressions, for all the details look at `?regex`.

Expressions in braces can be captured in `\\1`, `\\2`, etc

Some examples:

- `^` - beginning of a string
- `[A-Z0-9]` any capital letter or number
- `[^A-Z]` any character except capital letters (second meaning of the tegmentum)
- `.` (dot) any character except newline
- `[A-Z]{2}` two capital letters in sequence
- `[A-Z]+` one or any number of capital letters
- `[a-z]*` zero or any number of lowercase letters
- `[ ^ ]{2,5}` two to five letters which are not a space



## Task 4 - Write data into file

- We now like to write the tabular output into an output file.
- To the function `getCounts` add an optional argument `outfile` which defaults to an empty string.
- So something like `getCounts(filename, outfile="")`
- If no `outfile` is given we should still just write the output to the terminal.
- If an `outfile` is given we should write our data to this file.
- Add the implementation to write to this `outfile` to your function
- add check for a second argument to your application into your main function, if this argument exists, add it to the function call.

```
# within main function do this
args=commandArgs(trailingOnly=TRUE)
if (length(args) == 2) {
    getCounts(args[1], outfile=args[2])
} else if (length(args) == 1) {
```

```
    getCounts(args[1])  
} else {  
    help()  
}
```

## Homework studies:

- [https://www.tutorialspoint.com/r/r\\_functions.htm](https://www.tutorialspoint.com/r/r_functions.htm)
- [https://www.tutorialspoint.com/r/r\\_csv\\_files.htm](https://www.tutorialspoint.com/r/r_csv_files.htm)
- <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf> (Chapter 10 - writing functions)

## Tomorrow:

- ⇒ Object oriented programming!!
- ⇒ R base graphics system - let's paint!!

# Programming with R

## Day 3

### Object Oriented Programming

### Graphics

### University of Potsdam

Detlef Groth

# Last Lecture

- R function:
  - mandatory
  - optional
  - delegation
- File IO
  - read.table
  - file r, w
- command line arguments, commandArgs
- terminal applications, while(TRUE)

# 3 Object Oriented Programming

## Outline

3.1	Object Oriented Programming . . . . .	167
3.2	S3 . . . . .	172
3.3	Proto . . . . .	178
3.4	RDS files . . . . .	189
3.5	Code documentation . . . . .	192
3.6	R graphics system . . . . .	206
3.7	Exercise OOP . . . . .	221

# Outline

## Day 1 (basics)

- setup, install editor, simple programs
- variables, operators
- data structures, control flow

## Day 2 (basics)

- functions
- file input/output
- terminal interaction
- command line arguments

## Day 3 (advanced)

- **object oriented progr.**
- **code documentation**
- **R base graphics system**

## Day 4 (advanced)

- using packages
- writing packages
- package documentation

## Day 5 (advanced)

- graphical user interfaces
- tcltk (shiny)

# 3.1 Object Oriented Programming

## Object Oriented Programming:

... is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. OOP languages are diverse ...

[Wikipedia — The free Encyclopedia, 2020]

# Goal of OOP

- simulate real world actions
- create objects with methods and properties
- logic is naturally divided
- Example objects in a statistical analysis:
  - DataReader
    - \* read.data (patient data from database)
    - \* create new variables (count amino acid two letter codes, ...)
  - Plotter (amino acid sequences, amino acid statistics)
  - FastaUtils (read fasta, read FASTA data NCBI from website, ...)
  - ...



# Terminology

- Class: template for creating objects
- (Class variable: Variables shared by all objects of a class)
- (Class method: functions for the class)
- Object: a certain instance of a class
- Instance variable: a variable for each object (property)
- Instance method: functions working for the objects
- Inheritance: methods & properties inherited from other classes
- Overriding: methods can be overwritten by a child class to give them a new functionality

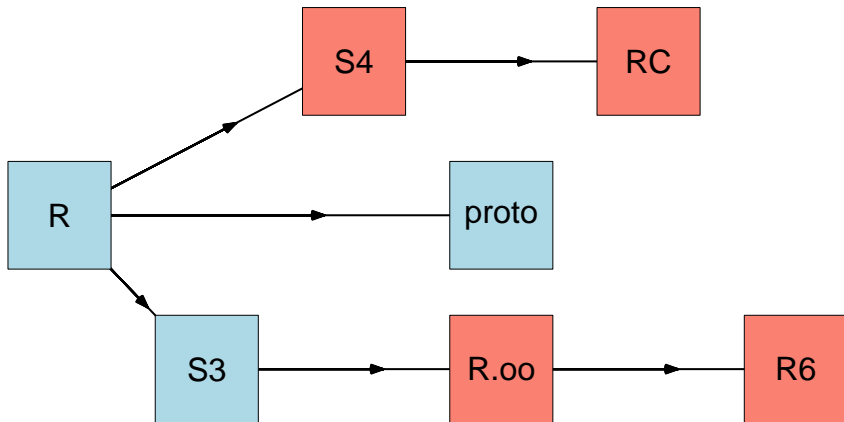
Most R class systems have no support for class variables and class methods. R is more object oriented in this case.

# R OOP systems

- **S3** (1986) - core - most often used
- **S4** (1992) - core - extension to S3
- **RC** (2010) - extension to S4 - core
- **R6** (2014) - extension to S3 - separate library
- **R.oo** (2005) - extension to S3 - separate library
- **proto** (2005) - very object oriented - separate library

⇒ we will discuss here **S3** as it is the most widely used OO system in R and **proto**, as this is quite close to the syntax of Python or Perl object oriented systems.

# OOP Evolution



## 3.2 S3

⇒ **Syntax:** `method(object, args)`

```
> options(continue='  ')\n> mike=list(name='Mike',type='animal')\n> class(mike)='Animal'\n> print(mike)
```

```
$name
```

```
[1] "Mike"
```

```
$type
```

```
[1] "animal"
```

```
attr(,"class")
```

```
[1] "Animal"
```

# Create a print.Animal function

```
> print.Animal <- function (self) {  
  print(paste('My name is ', self$name, '!', sep=''))  
}  
> print(mike)  
[1] "My name is Mike!"
```

That is ok! But what about let run Mike:

```
> run.Animal <- function (self) {  
  print(paste(self$name, ' runs!', sep=''))  
}  
> try (run(Mike), silent=FALSE, outFile=stdout())  
Error in run(Mike) : could not find function "run"  
> methods(class='Animal')  
[1] print  
see '?methods' for accessing help and source code
```

# Need a generic function - run.default

```
> run = function (x,...) UseMethod("run")
> run.default = function (x) {
  print('generic run')
}
> run('Hi')
[1] "generic run"
> run(521)
[1] "generic run"
> run(list(a=1,b=1:4))
[1] "generic run"
> run(mike)
[1] "Mike runs!"
> methods(class='Animal')
[1] print run
see '?methods' for accessing help and source code
```

# Inheritance with S3

```
> susi=list(name='Susi',age=5,sex='lady')
> class(susi)=c('Animal','Cat')
> run(susi)
[1] "Susi runs!"
```

That's nice, let's meow her:

```
> meow.Cat = function (self,n=2) {
  return(paste(self$name, ' says ', paste(rep('meow',n)
})
> meow = function (x,...) UseMethod("meow")
> meow.default = function (x,...) {
  print('generic meow')
}
> meow(susi,n=4)
[1] "Susi says meow, meow, meow, meow!"
```

# Disadvantages of S3

```
> class(susi)
[1] "Animal" "Cat"
> methods(class=class(susi))
[1] print run
see '?methods' for accessing help and source code
> for (cls in class(susi)) {
  print(methods(class=cls))
}
[1] print run
see '?methods' for accessing help and source code
[1] meow
see '?methods' for accessing help and source code
```

- fine for standard methods like print, plot, summary
- for more specific methods (meow) we need actually to declare three methods each time
- not so easy access to methods of the class



# Summary S3

- S3 use method dispatching
  - installation procedure:
    - 1) `methname = function (x, ...)`  
`UseMethod('methname')`
    - 2) create a default method: `methname.default`
    - 3) create `methname.Classname` for your class
- ⇒ hint use a snippet for this three step procedure
- if you would like to learn only one OOP system, you should probably learn S3

## 3.3 Proto

⇒ Syntax: `object$method(args)`

⇒ Proto works on objects.

```
> #install.packages('proto')
> library(proto)
> moritz=proto(name='Moritz',type='rat',age=1)
> ls(moritz)
[1] "age"  "name" "type"
> print(moritz$name)
[1] "Moritz"
> moritz$run = function (self) {
    paste(self$name, ' runs!', sep='')
}
> moritz$run()
[1] "Moritz runs!"
```

# The self argument

- first argument of proto method is the object itself
- you can name it as you like
- some people use a dot '.', other 'this'
- I prefer 'self' as self is used as well in Python and Perl by convention
- they took this name from the programming language Self (1986) which was one of the first object oriented languages
- Self, as JavaScript or proto are an OOP style based on prototypes
- prototype programming means we create simple objects first and then extend them stepwise as we need more features

# Don't forget the self

- it is often that we define the self in the method declaration
- a typical error with forgotten self looks like this

```
library(proto)
math=proto()
math$add <- function (x,y) { # incorrect, no self
  return(x+y)
}
math$add(1,3)
```

The error you get is:

```
Error in res(x, ...) : unused argument (3)
Calls: <Anonymous>
Execution halted
```

## Fix:

```
library(proto)
math=proto()
math$add <- function (self,x,y) { # corrected
  return(x+y)
}
math$add(1,3)
-> 4
```

**So calling** `math$add(1, 3)` **actually is calling**  
`math$add(math, 1, 3)`.

⇒ **Advantage:** The object itself is always available in the function body!

# Let's create first a class **Animal**, than an individual rat animal

```
> Animal = proto()
> Animal$new <- function (self,name, # constructor
  type='generic animal',
  age=0) {
  self$name=name
  self$type=type
  self$age=age
  self$.km=0 ; # hidden/private dot variable
  return(self) # return the object
}
> Animal$run <- function (self,km=1) {
  self$.km=self$.km+km
  return(paste(self$name, ' runs ',km,'!'))
}
> Animal$getKm <- function (self) { # standard method
  return(self$.km)
}
```

```

> ls(Animal)
[1] "getKm" "new"    "run"
> moritz=Animal$new(name='Moritz',type='Rat',age=1)
> moritz$run(km=0.1)
[1] "Moritz runs 0.1 !"
> moritz$run(km=0.5)
[1] "Moritz runs 0.5 !"
> moritz$getKm()
[1] 0.6
> # introspection
> ls(moritz)
[1] "age"    "getKm"  "name"   "new"    "run"    "type"
> str(moritz)
proto object
 $ getKm:function (self)
   ..- attr(*, "srcref")= 'srcref' int [1:8] 366 17 368 1
   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcf
 $ type : chr "Rat"

```

```
$ name : chr "Moritz"
$ age  : num 1
$ run  :function (self, km = 1)
  ..- attr(*, "srcref")= 'srcref' int [1:8] 362 15 365 1
  .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcf
$ new  :function (self, name, type = "generic animal", a
  ..- attr(*, "srcref")= 'srcref' int [1:8] 353 15 361 1
  .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcf
```



# Inheritance

```
> Cat = Animal$proto() # inheritance from Animal
> Cat$meow = function (self,n=2) {
  return(paste(self$name, ' says ',
    paste(rep('meow',n),collapse=', '), '!', sep=''))
}
> ls(Cat)
[1] "meow"
> susi=Cat$new('Susi',type='Cat')
> susi$meow(n=5)
[1] "Susi says meow, meow, meow, meow, meow!"
> ls(susi)
[1] "age" "meow" "name" "type"
> susi$run(km=1.1)
[1] "Susi runs 1.1 !"
```

```
> susi$getKm()
[1] 1.1
> class(susi)
[1] "proto"          "environment"
> susi$ls()
[1] "age"  "meow" "name" "type"
> str(Cat)
proto object
 $ meow: function (self, n = 2)
   ..- attr(*, "srcref")= 'srcref' int [1:8] 383 12 387 1
   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcf
 $ type: chr "Cat"
 $ name: chr "Susi"
 $ age : num 0
 parent: proto object
> ls(susi$.super)
[1] "age"      "getKm"    "name"     "new"      "run"      "type"
```

# Code conventions

- Classes: uppercase letters `FastaUtil = proto()`
- Objects: lowercase letters `fu = FastaUtil$proto()`
- public methods: lowercase letters  
`FastaUtil$read.fasta`
- private methods (used only within the class): starting with dot, `_` or `uc` letters: `FastaUtil$.seqcheck`
- Files: each class in its own file with the name of the class as basename `FastaUtil.R`
- separate folder for applications based on your classes
- `lib` for classes, `app` or `bin` for applications
- conventions are optional but very helpful for organizing your work und mandatory for larger projects especially if you cooperate with other persons on the same project

# Summary proto

- prototype based programming
- very flexible, easy dynamic extension of objects instead of using mostly static class templates like in R6 (R), Java or C++
- classes in proto are just objects as well
- the first automatic self argument allows access to the object within the function
- don't forget `self`
- Geany shows you as well the functions in the document outline (not the case with S4, R6)
- you inherit from existing objects using the `proto` method of them

## 3.4 RDS files

### saveRDS and readRDS

- Advantage of proto - you can save easily your objects with all properties and methods to the harddisk!

- Command to save:

```
saveRDS(object, file=filename)
```

- Command to read back:

```
object=readRDS(filename)
```

```
> susi$run(1.2)
[1] "Susi runs 1.2 !"
> susi$run(0.5)
[1] "Susi runs 0.5 !"
> susi$getKm()
[1] 2.8
> saveRDS(susi, file='susi.RDS')
```

# Object reloading - How cool is that!!

```
#!/usr/bin/env Rscript
library(proto)
susi2=readRDS('susi.RDS')
ls(susi2)
ls(susi2$.super)
susi2$getKm()
susi2$run(0.1)
susi2$getKm()
susi2$meow(n=1)
⇒ [1] "age" "meow" "name" "type"
⇒ [1] "age" "getKm" "name" "new" "run" "type"
⇒ [1] 2.8
⇒ [1] "Susi runs 0.1 !"
⇒ [1] 2.9
⇒ [1] "Susi says meow!"
```

 readrds.R

# RDS files

- `writeRDS/readRDS`: Functions to write a single R object to a file, and to restore it
- multiple object can be combined into a list then, the list will be stored
- complete objects with data and analysis functions can be stored to the harddisk and used by anyone at a later time
- imagine, you could send your data and your analysis function even as E-Mail attachment to other researchers
- `proto` supports this type of analysis, the other OO systems not (maybe R6)

## 3.5 Code documentation

- general comments (after #):
  - for the developer to explain in the code why certain things are done that way
  - TODO's what is still needs to be done, etc
- **documenting functions and showing use cases (#')**
  - to use your own functions in other use cases, you don't like to look always in your source code
  - have your own set of help pages which extract the important things for your own written code

⇒ Code documentation - major key for a good programmer!

⇒ General comments are much less important!



# RD documentation

⇒ RD file format recently the standard

⇒ latex like syntax written in separate files,

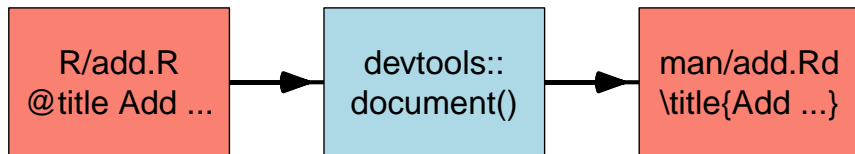
R/load.R ⇒ man/load.Rd

Example:

```
% File src/library/base/man/load.Rd
\name{load}
\alias{load}
\title{Reload Saved Datasets}
\description{
  Reload the datasets written to a file with the
  function
  \code{save}.
}
...
```

# Roxygen2 documentation

- newer approach
- documentation is directly added to source code
- lines with code documentation are starting with `#'`
- separate R commands are used to create then the Rd files out of the R files automatically
- `library(devtools)` function  
`devtools::document()` above the package directory



# Roxygen2 example

```
#' @title Illustration of crayon colors
# '
# ' @description Creates a plot of the crayon colors in
# '       \code{\link{brocolors}}
# '
# ' @param method2order method to order colors
# '       (\code{"hsv"} or \code{"cluster"})
# ' @param cex character expansion for the text
# ' @param mar margin parameters; vector of length 4
# '       (see \code{\link[graphics]{par}})
# '
# ' @return None
# '
# ' @examples
# ' plot_crayons()
# '
# ' @export
```

```
plot_crayons <-  
function(method2order=c("hsv", "cluster"), cex=0.6, mar=r  
{ ... }
```

# Practical illustration

```
#!/usr/bin/env Rscript
if (!require('devtools')) {
  install.packages('devtools')
}
usage = function () {
  cat('roxy.R converting R files comments to Rd markup\n')
  cat('Author: D. Groth, University of Potsdam, 2020\n')
  cat('Usage: roxy.R directory\n')
}
indir=commandArgs(trailingOnly=TRUE)
if (length(indir)==0) {
  usage()
} else {
  if (!dir.exists(indir[1])) {
    usage()
  } else {
```

```
    devtools::document(indir[1])  
  }  
}
```

⇒ roxy.R converting R files comments to Rd markup manual  
⇒ Author: D. Groth, University of Potsdam, 2020  
⇒ Usage: roxy.R directory



# Document in an interactive R-session

- use `setwd('parentpath')` to go into the parent directory of your package
- then call `devtools::document('pkgname')`

```
> setwd("rlibs-dev/")
> list.files()
[1] "asg"      "asgmd"    "asgui"
[4] "cat"      "dgtools"
> library(devtools)
> devtools::document('cat')
Updating cat documentation
Writing NAMESPACE
Loading cat
Welcome Package!
Writing NAMESPACE
```

# Create packages (pwr2020)

```
#!/usr/bin/R --vanilla --slave -f
if (!dir.exists('pwr2020')) {
  add <- function(x, y) {
    x + y
  }
  tdata=data.frame(a=1:10,b=rnorm(10),c=LETTERS[1:10])
  package.skeleton('pwr2020')
}
options(width=50)
list.files('pwr2020',recursive=TRUE)
file.show('pwr2020/R/add.R')
⇒ [1] "data/tdata.rda"
⇒ [2] "DESCRIPTION"
⇒ [3] "LICENSE"
⇒ [4] "man/add.Rd"
⇒ [5] "man/add.Rd~"
⇒ [6] "man/pwr2020-package.Rd"
```



```
⇒ [7] "man/tdata.Rd"
⇒ [8] "NAMESPACE"
⇒ [9] "R/add.R"
⇒ [10] "R/add2.R"
⇒ [11] "R/pwr2020-internal.R"
⇒ [12] "R/pwr2020-internal.R~"
⇒ [13] "Read-and-delete-me"
⇒ add <-
⇒ function (x, y)
⇒ {
⇒     x + y
⇒ }
⇒
```

 pkgskel.R

⇒ more about packages on Day 4

# Create add2.R with Roxygen2 documentation

```
> if (file.exists('pwr2020/man/add2.Rd')) {  
  file.remove('pwr2020/man/add2.Rd')  
}  
[1] TRUE  
> cat("  
#' Add together two numbers  
#'  
#' @param x A number  
#' @param y A number  
#' @return The sum of \\code{x} and \\code{y}  
#' @examples  
#' add2(1, 1)  
#' add2(10, 1)  
add2 <- function(x, y) {  
  x + y  
}  
", file='pwr2020/R/add2.R')
```

# Creating add2.Rd file

```
#!/bin/sh
ls pwr2020/man
./roxy.R pwr2020
ls pwr2020/man
head pwr2020/man/add2.Rd
⇒ add.Rd
⇒ add.Rd~
⇒ pwr2020-package.Rd
⇒ tdata.Rd
⇒ Writing add2.Rd
⇒ add2.Rd
⇒ add.Rd
⇒ add.Rd~
⇒ pwr2020-package.Rd
⇒ tdata.Rd
⇒ % Generated by roxygen2: do not edit by hand
⇒ % Please edit documentation in R/add2.R
```

```
⇒ \name{add2}  
⇒ \alias{add2}  
⇒ \title{Add together two numbers}  
⇒ \usage{  
⇒ add2(x, y)  
⇒ }  
⇒ \arguments{  
⇒ \item{x}{A number}
```



# Code documentation summary

- for creating help pages of your code
- do this even if you are the only user of your code
- embedding documentation into source code is best
- commenting using Roxygen2 tags directly above the code
- if you change arguments to your function, just update the docs before
- function to create documentation: `devtools::document`
- more about packages tomorrow

## 3.6 R graphics system

- graphics - core
- lattice - core
- ggplot2 - additional external popular library

⇒ we will only cover standard graphics here

⇒ it is easy to use for beginners

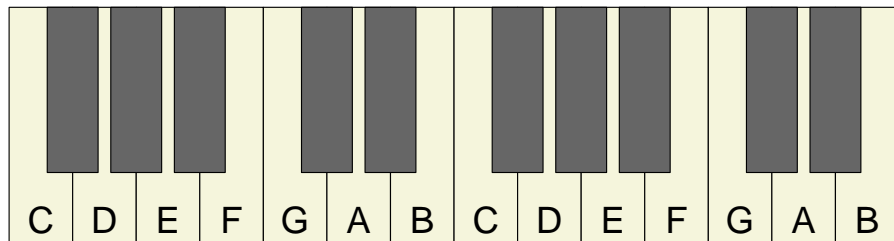
⇒ it can be used comfortably as well for the experienced programmer even it is sometimes less flexible

⇒ with lattice and ggplot, graphics can be variables

⇒ with base graphics we just paint on the graphics device

# Example 1

```
> par(mai=rep(0.1, 4))
> # empty painting surface
> plot(1,type='n',xlab='',ylab='',
      xlim=c(1,15),ylim=c(0,3),axes=FALSE)
> for (i in 1:14) {
  rect(i,0.25,i+1,2.75,col='beige')
  text(i+0.5,0.5,
       label=rep(c('C','D','E','F','G','A','B'),2)[i],
       cex=2)
}
> for (i in c(1,2,3,5,6,8,9,10,12,13)) {
  rect(i+0.6,1,i+1.4,2.75,col='grey40')
}
```





## Example 2

```
> # empty painting surface
> par(mai=rep(0.1, 4))
> plot(1, type='n', xlab='', ylab='',
       xlim=c(0, 2), ylim=c(0, 3), axes=FALSE)
> labels=rev(c('Thu', 'Fri', 'Wknd', 'Mon', 'Tue', 'Wed'))
> topics=c('OOP', 'Functions', 'Intro', 'Relaxing ...',
           'GUI', 'Packages')
> z=1
> for (x in 0:1) {
  for (y in 0:2) {
    rect(x, y, x+1, y+1, lwd=3)
    rect(x, y, x+0.2, y+0.2, lwd=1)
    for (y2 in 1:5) {
      lines(c(x, x+1), c((y+y2*0.2), c(y+y2*0.2)))
    }
    text(x+0.1, y+0.1, label=labels[z])
  }
}
```

```
    text(x+0.5, y+0.5, label=topics[z], cex=2,  
         family="serif")  
    z=z+1  
  }  
}
```

Intro		Packages	
Mon		Thu	
Functions		GUI	
Tue		Fri	
OOP		Relaxing ...	
Wed		Wknd	

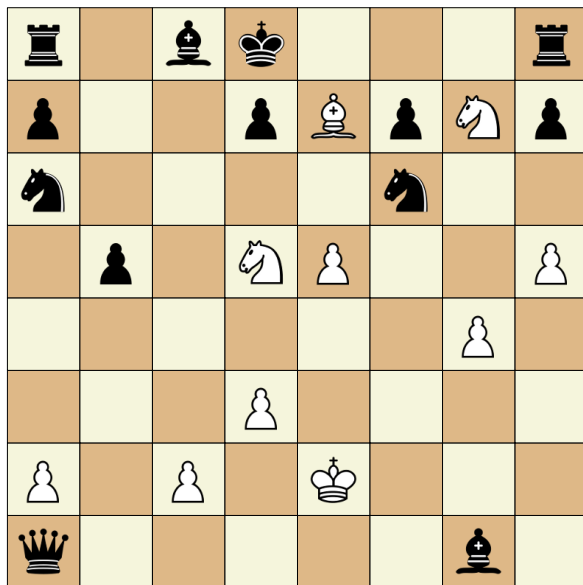
## Example 3

```
> library(png)
> par(mai=rep(0.1, 4))
> plot(1, type='n', xlab='', ylab='',
      xlim=c(0, 8), ylim=c(0, 8), axes=FALSE)
> imgDir='/home/groth/workspace/delfgroth/mytcl/dchess/'
> fen='r1bk3r/p2pBpNp/n4n2/lp1NP2P/6P1/3P4/P1P1K3/q5b1'
> for (x in 1:8) {
    fen=gsub(x, paste(rep(' ', x), collapse=' '), fen)
}
> fen=gsub('/ ', ' ', fen)
> cols=rep(c(rep(c('burlywood', 'beige'), 4),
             rep(c('beige', 'burlywood'), 4)), 4)
> k=1
> imgdir='/home/groth/workspace/delfgroth/mytcl/dchess/wi
> for (i in 0:7) {
    for (j in 0:7) {
        rect(i, j, i+1, j+1, col=cols[k])
```

```

        k=k+1
    }
}
> for (i in 0:7) {
  for (j in 0:7) {
    idx=64-8*(i+1)+j+1
    piece=substr(fen,idx,idx)
    if (piece != ' ') {
      if (grepl('[A-Z]',piece)) {
        col='w' } else { col='b' }
      imgfile=paste(imgdir,col,toupper(piece),
        '.png',sep='')
      pic <- readPNG(imgfile)
      rasterImage(pic, xleft = j+0.1,
        ybottom = i+0.1,
        xright=j+0.9,ytop=i+0.9)
    }
  }
}

```



# R plotting - devices

- screen
  - x11() - Unix
  - windows() - Windows
  - quartz() - Mac OSX
- files
  - pdf (vector)
  - svg (vector)
  - png (bitmap)
  - jpeg (bitmap)
  - ...
- close device - dev.off()

# PDF device

- the preferential device for publications
- multipage enabled
- options
  - width (inches)
  - height (inches)

```
> args(pdf)
```

```
function (file = if (onefile) "Rplots.pdf" else "Rplot",  
  width, height, onefile, family, title, fonts, vectorFonts,  
  encoding, bg, fg, pointsize, pagecentre, colormodel, compress,  
  useKerning, fillOddEven, compress)
```



# Multifigure plots

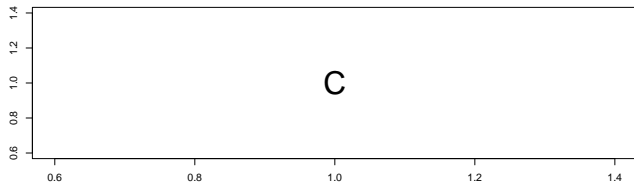
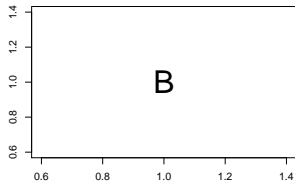
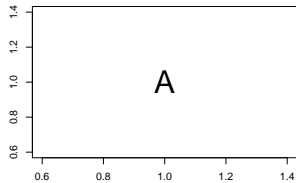
- `par(mfrow=c(2,3))` - two rows, three columns with rowwise filling
- `par(mfcol=c(2,3))` - two rows, three columns with colwise filling
- `layout()` - different arrangements possible

```
> pdf('test.pdf', width=9, height=6)
> par(mai=rep(0.5, 4))
> mt=matrix(c(1, 2, 3, 3), nrow=2, byrow=TRUE)
> mt
```

	[, 1]	[, 2]
[1, ]	1	2
[2, ]	3	3

```
> layout(mt)
> plot(1, pch='A', cex=3)
> plot(1, pch='B', cex=3)
> plot(1, pch='C', cex=3)
> dev.off()
```

null device  
1



# Summary graphics

- beside standard statistics plots we can build our own graphics
- flexible approaches are starting with an empty surface
- low level functions are used thereafter to plot different graphical elements like text, rectangles, images
- flexible multifigure graphics can be build using par settings or the layout function
- the preferred output for publication is PDF, sometimes with many, many data points bitmap graphics might be a better choice

# Lecture summary

## OOP:

- S3 (1986) - method(object)
- proto (2005) - object\$method()
- proto objects can be saved to the file system
- follow your code conventions

## Code documentation:

- document your functions, methods
- Roxygen2 for embedding docu beside the code

## Plotting:

- flexible base graphics package
- multiframe plots
- preferred output pdf

## 3.7 Exercise OOP

### Task 0 - prepare workspace and editor

- Start your Geany text editor.
- If not done yet download the FASTA files for the two types of Corona virus from the Moodle course site.
- Place the files under a data sub directory of your R files.
- More features of the Geany text editor:
  - Activate the Plugin File-Browser using the Menu Tools->Plugin manager.
  - On the left you should now have a file browser tab.
  - Templates are starting files for your programming.
  - Create an R template file app.R within the folder ~  
/.config/geany/templates/files/ (Unix) or on  
something like C:\Users\UserName\Roaming\geany  
(Windows).
  - To find out the correct path look at the beginning of the menu

- point output of 'Help->Debug Messages'.
- Please note that to save the new template you might activate the show hidden files option at the bottom of the file open dialog (Show more options).
  - For details about templates look later here at the documentation: <https://www.geany.org/manual/current/index.html#templates>.
  - The app.R template should have a shebang, the usage function, the main function.
  - After saving the file in the template directory reload the configuration (Tools->Reload Configuration) and the new file appears again.

# Task 1 - FastaUtil.R - outline and documentation

- Let us write a S3 class FastaUtil.
- At first we create a function read.fasta in the file FastaUtil.R.
- You can copy the read file functionality from yesterdays exercise into this new file.
- The function should have the following arguments:
  - infile: input filename
  - n: number of sequences to read from the file, if n == -1 all sequences should be read
  - id: possible id to be read from the file, then only teh sequence for this id is returned
- The function should return a list object of class FastaUtil.
- Test the code with both corona FASTA files available in Moodle.
- See below for an outline:

```

#' @title read sequences from a fasta file
#'
#' @description
#'
#' @param infile input filename of a standard
#'               fasta file
#' ...
#' @export
read.fasta <- function (infile,n=-1,id='') {
  fasta=list()
  # loop over file
  # loop over the file and create list entries for
  # each ID entries consisting of the id as the key
  # and a nested key with description and sequences
  # ex; l[[key]]=list(description="Hello description",
  #                   sequence="MATCL...",length=NN)
  class(fasta)="FastaUtil"
  return(fasta)
}

```



## Task 2 - FastaUtil.R - summary.FastaUtil

- R summary functions usually display information about the sequences.
- It should display how many sequences are stored in this object and then give a table about the length of each sequence.
- We loop over each key names(fasta) and we extract the relevant information to the terminal.
- We create a data frame within this function and return it.
- See below for an outline

```
#' @title summarize a FastaUtil sequence object
#' ...
summary.FastaUtil <- function (x) {
  names=names(x)
  lengths=c()
  for (nm in names) {
    # calculate lengths as nlength
    lengths=c(lengths,nlength)
  }
  return(data.frame(id=names,length=lengths))
}
```

# Homework:

- Have a short look at this R manual <https://cran.r-project.org/doc/manuals/R-lang.pdf> chapter 4 and 5
- Have a look at the proto Vignette:  
<https://cran.r-project.org/web/packages/proto/vignettes/proto.pdf>
- Have a look at the biophysical properties of amino acids  
<https://www.sigmaaldrich.com/life-science/metabolomics/learning-center/amino-acid-reference-chart.html>

# Programming with R

## Day 4

### Packages

### Using and Writing

## University of Potsdam

Detlef Groth

# Last Lecture

- R OOP:
  - OOP terminology
  - S3 method(object)
  - proto object\$method()
  - saveRDS/readRDS
- code documentation
  - R vs Rd files
  - roxygen2 documentation
- R base graphics:
- empty plot surface
- low level commands
- multiframe plots

# Outline

## Day 1 (basics)

- setup, install editor, simple programs
- variables, operators
- data structures, control flow

## Day 2 (basics)

- functions
- file input/output
- terminal interaction
- command line arguments

## Day 3 (advanced)

- object oriented progr.
- code documentation
- R base graphics system

## Day 4 (advanced)

- **using packages**
- **writing packages**
- **package documentation**

## Day 5 (advanced)

- graphical user interfaces
- tcltk (shiny)

# 4 R Packages

## Outline

4.1	Installing packages from the web . . . . .	233
4.2	Using packages . . . . .	256
4.3	Writing packages . . . . .	260
4.4	Exercise 4 - packages . . . . .	288

# Outline

- installation
  - CRAN
  - Bioconductor
- usage
  - library
  - require
- writing
  - coding
  - documenting
  - testing
  - packaging
  - installing



## 4.1 Installing packages from the web

- there are many R packages in the web available
- repositories (tested and quality controlled):
  - `https://cran.r-project.org/web/packages`
  - `https://www.bioconductor.org`
- from github (less trustable)

# CRAN

<https://cran.r-project.org/web/packages/>

- August 2009 - 1928 packages
- Juni 2011 - 3023 packages
- July 2012 - 3914 packages
- May 2013 - 4526 packages
- January 2015 - 6221 packages
- October 2016 - 9332 packages
- July 2017 - 10875 packages
- August 2018 - 12954 packages
- Juny 2019 - 14311 packages

# CRAN September 2020



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

## Contributed Packages

### Available Packages

Currently, the CRAN package repository features 16322 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

### Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual [R Installation and Administration](#) (also contained in the R base sources) explains the process in detail.

[CRAN Task Views](#) allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 41 views are available.

### Package Check Results


All packages are tested regularly on machines running [Debian GNU/Linux](#), [Fedora](#), macOS (formerly OS X), Solaris and Windows.

⇒ more than 16.000 packages

# 41 CRAN Task Views - 1

https://cran.r-project.org

For general concerns regarding task views contact the [CRAN](#) package maintainers.



**CRAN**  
[Mirrors](#)  
[What's new?](#)  
[Task Views](#)  
[Search](#)

**About R**  
[R Homepage](#)  
[The R Journal](#)

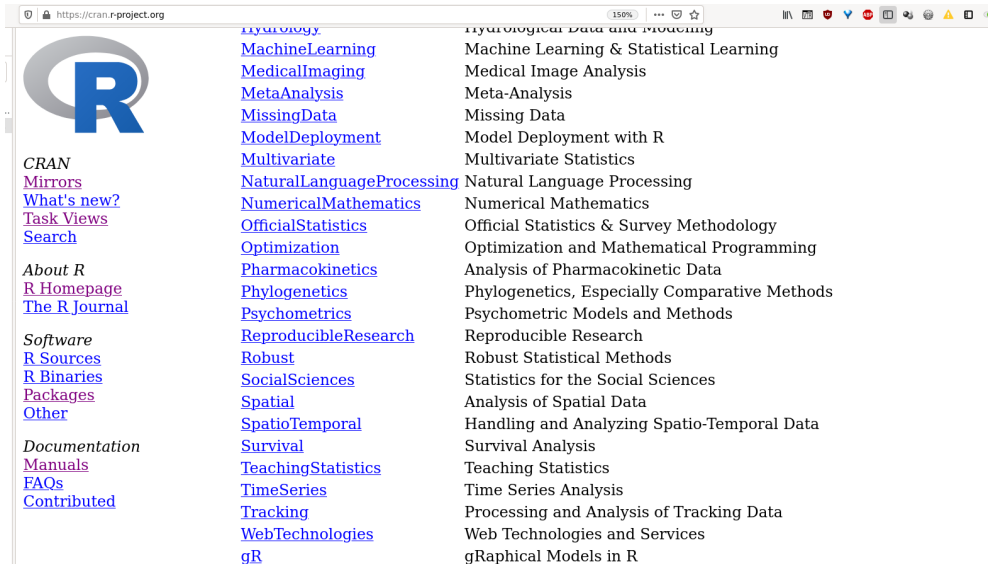
**Software**  
[R Sources](#)  
[R Binaries](#)  
[Packages](#)  
[Other](#)

**Documentation**  
[Manuals](#)  
[FAQs](#)  
[Contributed](#)

**Topics**

<a href="#">Bayesian</a>	Bayesian Inference
<a href="#">ChemPhys</a>	Chemometrics and Computational Physics
<a href="#">ClinicalTrials</a>	Clinical Trial Design, Monitoring, and Analysis
<a href="#">Cluster</a>	Cluster Analysis & Finite Mixture Models
<a href="#">Databases</a>	Databases with R
<a href="#">DifferentialEquations</a>	Differential Equations
<a href="#">Distributions</a>	Probability Distributions
<a href="#">Econometrics</a>	Econometrics
<a href="#">Environmetrics</a>	Analysis of Ecological and Environmental Data
<a href="#">ExperimentalDesign</a>	Design of Experiments (DoE) & Analysis of Experimental Data
<a href="#">ExtremeValue</a>	Extreme Value Analysis
<a href="#">Finance</a>	Empirical Finance
<a href="#">FunctionalData</a>	Functional Data Analysis
<a href="#">Genetics</a>	Statistical Genetics
<a href="#">Graphics</a>	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
<a href="#">HighPerformanceComputing</a>	High-Performance and Parallel Computing with R
<a href="#">Hydrology</a>	Hydrological Data and Modeling
<a href="#">MachineLearning</a>	Machine Learning & Statistical Learning
<a href="#">MedicalImaging</a>	Medical Image Analysis
<a href="#">MetaAnalysis</a>	Meta-Analysis
<a href="#">MissingData</a>	Missing Data

# 41 CRAN Task Views - 2



The screenshot shows the CRAN Task Views page. The browser address bar displays 'https://cran.r-project.org'. The page features the CRAN logo (a stylized 'R' inside a circle) and a list of links categorized under 'CRAN', 'About R', 'Software', and 'Documentation'. The 'CRAN' category includes links to 'Mirrors', 'What's new?', 'Task Views', and 'Search'. The 'About R' category includes links to 'R Homepage' and 'The R Journal'. The 'Software' category includes links to 'R Sources', 'R Binaries', 'Packages', and 'Other'. The 'Documentation' category includes links to 'Manuals', 'FAQs', and 'Contributed'. The main content area lists 41 task views, each with a link to its corresponding page. The task views are: Hydrology, Machine Learning, Medical Imaging, Meta-Analysis, Missing Data, Model Deployment, Multivariate, Natural Language Processing, Numerical Mathematics, Official Statistics, Optimization, Pharmacokinetics, Phylogenetics, Psychometrics, Reproducible Research, Robust, Social Sciences, Spatial, Spatio-Temporal, Survival, Teaching Statistics, Time Series, Tracking, Web Technologies, and gR.

CRAN

- [Mirrors](#)
- [What's new?](#)
- [Task Views](#)
- [Search](#)

About R

- [R Homepage](#)
- [The R Journal](#)

Software

- [R Sources](#)
- [R Binaries](#)
- [Packages](#)
- [Other](#)

Documentation

- [Manuals](#)
- [FAQs](#)
- [Contributed](#)

[Hydrology](#)

[Machine Learning](#)

[Medical Imaging](#)

[Meta-Analysis](#)

[Missing Data](#)

[Model Deployment](#)

[Multivariate](#)

[Natural Language Processing](#)

[Numerical Mathematics](#)

[Official Statistics](#)

[Optimization](#)

[Pharmacokinetics](#)

[Phylogenetics](#)

[Psychometrics](#)

[Reproducible Research](#)

[Robust](#)

[Social Sciences](#)

[Spatial](#)

[Spatio-Temporal](#)

[Survival](#)

[Teaching Statistics](#)

[Time Series](#)

[Tracking](#)

[Web Technologies](#)

[gR](#)

Hydrological Data and Modeling

Machine Learning & Statistical Learning

Medical Image Analysis

Meta-Analysis

Missing Data

Model Deployment with R

Multivariate Statistics

Natural Language Processing

Numerical Mathematics

Official Statistics & Survey Methodology

Optimization and Mathematical Programming

Analysis of Pharmacokinetic Data

Phylogenetics, Especially Comparative Methods

Psychometric Models and Methods

Reproducible Research

Robust Statistical Methods

Statistics for the Social Sciences

Analysis of Spatial Data

Handling and Analyzing Spatio-Temporal Data

Survival Analysis

Teaching Statistics

Time Series Analysis

Processing and Analysis of Tracking Data

Web Technologies and Services

gRaphical Models in R

# Installation from CRAN

On Unix systems with package managers, try first the package manager, this ensures automatic updates.

Linux Fedora:

```
[groth@bariuke build]$ sudo dnf search R-devtools
[sudo] password for groth:
Last metadata expiration check: 9:59:20 ago on Sun
  20 Sep 2020 07:33:48 PM CEST.
===== Name Exactly Matched: R-devtools =====
R-devtools.noarch : Tools to Make Developing R
  Packages Easier

[groth@bariuke build]$ sudo dnf install R-devtools
...
```

# Fedora how many packages?

```
[groth@bariuke build]$ sudo dnf search R-* | head
Last metadata expiration check: 10:00:36 ago on
  Sun 20 Sep 2020 07:33:48 PM CEST.
```

```
===== Name & Summary Matched: R-* =====
```

```
R-littler-examples.x86_64 : R-littler Examples
```

```
===== Name Matched: R-* =====
```

```
R-ALL.noarch : Data of T- and B-cell Acute Lymphocytic
               Leukemia
```

```
R-AUC.noarch : Threshold independent performance measures
               for probabilistic classifiers
```

```
R-AnnotationDbi.noarch : Annotation Database Interface
```

```
R-AsioHeaders-devel.noarch : Asio C++ Header Files
```

```
R-BH-devel.noarch : Boost C++ Header Files for R
```

```
R-BSgenome.noarch : Infrastructure for Biostrings-based
                   genome data packages
```

```
R-BSgenome.Celegans.UCSC.ce2.noarch : Caenorhabditis
                                     elegans genome (UCSC Release ce2)
```

```
[groth@bariuke]$ sudo dnf search R-* | grep -E '^R-' |  
    grep -v i686 | wc -l
```

Last metadata expiration check: 10:01:12 ago on  
Sun 20 Sep 2020 07:33:48 PM CEST.

414



# Information about a package

```
[groth@bariuke build]$ sudo dnf info R-brio
```

## Available Packages

Name	: R-brio
Version	: 1.1.0
Release	: 1.fc31
Architecture	: x86_64
Size	: 42 k
Source	: R-brio-1.1.0-1.fc31.src.rpm
Repository	: updates
Summary	: Basic R Input Output
URL	: <a href="https://CRAN.R-project.org/package=brio">https://CRAN.R-project.org/package=brio</a>
License	: MIT
Description	: Functions to handle basic input output, : these functions always read and write : UTF-8 (8-bit Unicode Transf. Format) : files and provide more explicit : control over line endings

# Installation within R

```
install.packages ( 'pkgname' )
```

Other commands:

- `available.packages`
- `update.packages`
- `remove.packages`

# Package dependencies

If you have the choice between different packages which solves your task prefer packages with few dependencies over packages with many and prefer the MIT or BSD license over GPL licenses. GPL license requires that you always publish your source code if you add such packages to your application bundle.

```
> options(continue=' ')
> options(width=55)
> package.deps <- function(x,mode='all') {
  if (!interactive()) {
    r <- getOption("repos");
    r["CRAN"] <- "https://lib.ugent.be/CRAN/"
    options(repos=r)
  }
}
```

```

require(tools)
deps=package_dependencies(x,recursive=TRUE)[[1]]
if (mode == 'install') {
  idx = which(
    !(deps %in% rownames(installed.packages()))
  )
  return(deps[idx])
} else if (mode == 'nonbase') {
  ipacks=installed.packages()
  bpacks=ipacks[ipacks[, 'Priority'] %in%
    c('base', 'recommended'), ]
  rnms=setdiff(rownames(ipacks), rownames(bpacks))
  return(intersect(deps, rnms))
} else if (mode == 'all') {
  return(deps)
} else {
  stop('mode is either install or nonbase')
}
}

```

```
> package.deps('igraph',mode='all')
[1] "methods"      "graphics"      "grDevices"     "magrittr"
[5] "Matrix"       "pkgconfig"     "stats"         "utils"
[9] "grid"         "lattice"

> package.deps('igraph',mode='nonbase')
[1] "magrittr"      "pkgconfig"

> package.deps('argparse',mode='nonbase')
[1] "R6"            "jsonlite"

> package.deps('argparser',mode='nonbase')
character(0)
```

# packageDescription

```
> packageDescription('argparser')
```

```
Package: argparser
```

```
Type: Package
```

```
Title: Command-Line Argument Parser
```

```
Version: 0.4
```

```
Date: 2016-04-03
```

```
Author: David J. H. Shih
```

```
Maintainer: David J. H. Shih  
<djh.shih@gmail.com>
```

```
Description: Cross-platform command-line  
argument parser written purely in R with no  
external dependencies. It is useful with  
the Rscript front-end and facilitates  
turning an R script into an executable  
script.
```

```
URL: https://bitbucket.org/djhshih/argparser
```

BugReports:

<https://bitbucket.org/djhshih/argparser/issues>

Depends: methods

License: GPL ( $\geq 3$ )

RoxygenNote: 5.0.1

NeedsCompilation: no

Packaged: 2016-04-04 00:02:25 UTC; davids

Repository: CRAN

Date/Publication: 2016-04-04 08:37:01

Built: R 3.6.1; ; 2019-11-03 13:22:01 UTC; unix

-- File: /usr/share/R/library/argparser/Meta/package.rds

```
> packageDescription('argparser')$License
```

```
[1] "GPL ( $\geq 3$ )"
```

But not showing non-installed packages:

```
> packageDescription('brio')
```

```
[1] NA
```

## url.show?

```
url.show("https://cran.r-project.org/web/packages/brio/")
```

⇒ But display to the terminal as html, unreadable



# url2txt

```
> url2txt <- function (url, file=tempfile(), quiet=TRUE) {  
  download.file(url, destfile = file, mode = "w",  
    quiet=quiet)  
  fin=fin = file(file, 'r')  
  res=''  
  flag=FALSE  
  while(length((line=readLines(fin, n=1L)))>0) {  
    if (grepl('<body>', line)) {  
      flag=TRUE  
    } else if (flag) {  
      line=gsub('</?.+?>', '', line)  
      res=paste(res, '\n', line)  
    }  
  }  
  close(fin)  
  return(res)  
}
```

```
> cat(substr(  
  url2txt("https://cran.r-project.org/web/packages/brio")  
  1,250))
```


brio: Basic R Input Output

Functions to handle basic input output, these functions read and write UTF-8 (8-bit Unicode Transformation Format) for more explicit control over line endings.

Version:

1.1.0

# Bioconductor



www.bioconductor.org

133%

Search:

[Home](#) [Install](#) [Help](#) [Developers](#) [About](#)

## About Bioconductor

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development. It has two releases each year, and an active user community. Bioconductor is also available as an [AMI](#) (Amazon Machine Image) and [Docker](#) images.

## News

- Bioconductor [3.12](#) release schedule is announced.
- [BIOCAsia](#) virtual conference registration is open (free registration!). October 15-18, 2020.
- [BIOEurope](#) virtual conference registration and abstract submission open December 14-18, 2020.
- See our [google calendar](#) for events, conferences, meetings, forums, etc. Add your event with email to events at

## Install »

- Discover [1903 software packages](#) available in Bioconductor release 3.11.

Get started with Bioconductor

- [Install Bioconductor](#)
- [Get support](#)
- [Latest newsletter](#)
- [Follow us on twitter](#)
- [Install R](#)

## Learn »

Master Bioconductor tools

- [Courses](#)
- [Support site](#)
- [Package vignettes](#)
- [Literature citations](#)
- [Common work flows](#)
- [FAQ](#)
- [Community resources](#)
- [Videos](#)

## Use »

Create bioinformatic solutions with Bioconductor

- [Software](#), [Annotation](#), and [Experiment](#) packages
- [Docker](#) and [Amazon](#) machine images
- Latest [release announcement](#)
- Use Bioconductor in the [AnVIL](#). See our [project updates](#).

## Develop »

Contribute to Bioconductor

- [Developer resources](#)
- [Use Bioc 'devel'](#)
- ['Devel' packages](#)
- [Package guidelines](#)
- [New package submission](#)
- [Git source control](#)
- [Build reports](#)

# Aim

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development. It has two releases each year, and an active user community.

...

Discover 1903 software packages available in Bioconductor release 3.11.

<https://bioconductor.org/>

# Installation of Bioconductor packages

Example: package impute - Imputation for microarray data  
(currently KNN only)

```
if (!requireNamespace("BiocManager",  
  quietly = TRUE)) {  
  # install from CRAN  
  install.packages("BiocManager")  
}  
# install from Bioconductor  
BiocManager::install("impute")
```

# CRAN vs Bioconductor

Bioconductor is more restrictive:

- should use S4 as OOP system
- S3 is not allowed
- must have vignette
- no line in code wider than 80 characters
- no dots in function or method names
- ...

See:

<http://bioconductor.org/developers/package-guidelines/>

# Github

Sometimes a package might be available only on github or you might (need) to try out the newest package version.

Example:

```
library(devtools)  
install_github("hadley/dplyr")
```

⇒ **Do this if you must!** No real integrity check for the code.

## 4.2 Using packages

- `library` - returns error if not installed
- `require` - returns false if not installed

```
> print(library(argparser)) # show which pkg is loaded
[1] "argparser" "tools"      "stats"      "graphics"
[5] "grDevices" "utils"      "datasets"   "methods"
[9] "base"

> print(require(argparse))
[1] FALSE

> print(require(argparser))
[1] TRUE

> print(require(argparsers))
[1] FALSE
```



# Use require for install on request

```
if (!require('argparser')) {  
  # in scripts you need to set the repo  
  if (!interactive()) {  
    r <- getOption("repos");  
    r["CRAN"] <- "https://lib.ugent.be/CRAN/"  
    options(repos=r)  
  }  
  install.packages('argparser')  
  library('argparser')  
}
```

# The `install.packages lib` argument

R has normally two standard library folders where it puts its R packages, it is recommend to place non-standard packages not into the main package folder (Windows: 'C:/Program Files/...').

So I usually use if I install as root the second standard folder, on Windows this is mostly somewhere in C:/Users.

To find out which are the folders where your R-libraries are places use the `.libPaths()` function:

```
> print(.libPaths())  
[1] "/home/groth/workspace/delfgroth/myr/rlibs"  
[2] "/home/groth/R/x86_64-redhat-linux-gnu-library/3.6"  
[3] "/usr/lib64/R/library"  
[4] "/usr/share/R/library"
```

# Modifying the .libPaths() folders

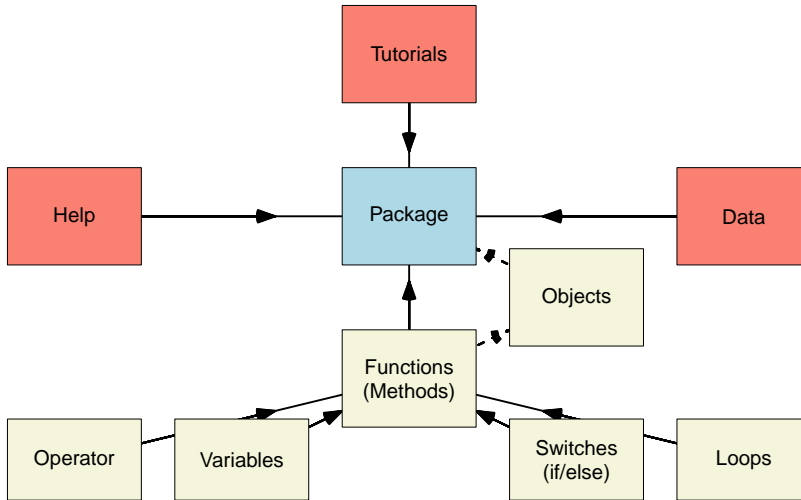
You can add manually an additional folder the .libPaths() folders.

```
.libPaths(c('/new/path/', .libPaths()))  
# now install it into the first folder of libPaths()  
install.packages('pkgname', lib=.libPaths()[1])
```

To use this new folder you have then every time to start your scripts with:

```
.libPaths(c('/new/path/', .libPaths()))
```

## 4.3 Writing packages



# THE References

R Core team Writing R extensions (199 pages!):

<https://cran.r-project.org/doc/manuals/r-release/R-exts.pdf>

Wickham - R packages (198 pages, Book):

<https://r-pkgs.org/> (HTML)

Take this as references but we try to make it shorter here (KISS).

# When to write a package?

When to start writing an R package?

As soon as you have 2 functions.

Why 2? After you have more than one function it starts to get easy to lose track of what your functions do, it starts to be tempting to name your functions `foo` or `tempfunction` or some other such nonsense. You are also tempted to put all of the functions in one file and just source it. That was what I did with my first project, which ended up being an epically comical set of about 3,000 lines of code in one R file. Ask my advisor about it sometime, he probably is still laughing about it.

<https://github.com/jtleek/rpackages>

## Benefits:

- more structured work
- you don't have to hunt for the latest version of a certain function
- you can publish it
- someone else can use it more easily than your arbitrary collections of code
- you can give him/her your tar.gz file of your package
- BTW: Where was this `package.deps` function or was it named `package.dependencies` ?

# Hints

- Write your first package if you start your first real project using R
- You might not have the intension to publish it at all, that is not the point, the point is to structure your work!
- Have one personal collection of functions in your own package (my one is `dgtools`) and one project specific package (I have `mcgraph`, `asg`, `swiss`, ...)
- use git to track your code changes
- we cover this in 1st semester course “Databases and Practical Programming (Using Python 3)”
- Good: 2nd semester (Machine Learning course)
- Latest: project thesis 3rd semester if done with R



# Steps

- setup a minimal R-package one function, one data set
- have on folder for all your developer versions of the package *rlibs-dev*
- run R CMD check *pkgname*
- run R CMD BUILD *pkgname*
- run R CMD INSTALL *pkgname.VERSION.tar.gz*
- have one folder for all your own installed packages *rlibs*

# KISS with devtools?

```
> package.deps('devtools', mode='nonbase')  
[1] "usethis"      "callr"        "cli"  
[4] "desc"         "ellipsis"     "httr"  
[7] "jsonlite"     "memoise"      "pkgbuild"  
[10] "pkgload"      "rcmdcheck"    "remotes"  
[13] "rlang"        "roxygen2"     "rstudioapi"  
[16] "sessioninfo"  "testthat"     "withr"  
[19] "processx"     "R6"           "assertthat"  
[22] "crayon"       "glue"         "fansi"  
[25] "digest"       "yaml"         "rprojroot"  
[28] "htmltools"    "htmlwidgets" "magrittr"  
[31] "crosstalk"    "promises"     "curl"  
[34] "mime"         "openssl"      "prettyunits"  
[37] "xopen"        "brew"         "commonmark"  
[40] "knitr"        "purrr"        "Rcpp"  
[43] "stringi"      "stringr"      "xml2"
```

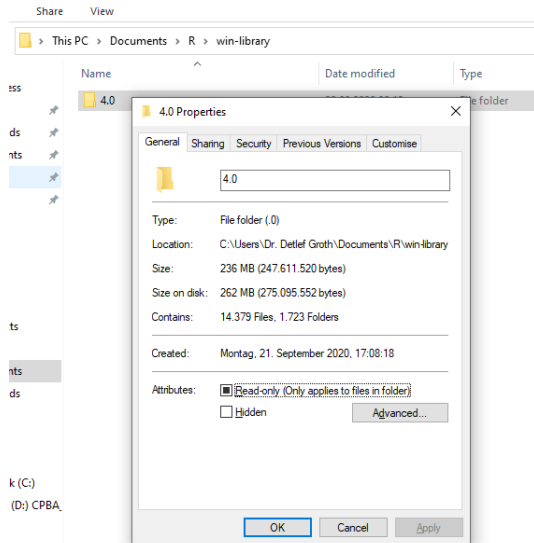
```
[46] "evaluate"      "praise"      "clipr"  
[49] "fs"            "gh"          "git2r"  
[52] "whisker"       "lazyeval"    "ini"  
[55] "base64enc"     "highr"       "markdown"  
[58] "xfun"          "askpass"     "ps"  
[61] "later"        "tibble"      "backports"  
[64] "sys"           "BH"          "lifecycle"  
[67] "pillar"       "pkgconfig"   "vctrs"  
[70] "utf8"
```

```
> length(package.deps('devtools', mode='nonbase'))
```

```
[1] 70
```

⇒ not really KISS

# 250MB, 14.000 files for devtools ...



# KISS with roxygen2? Only slightly better!

```
> package.deps('roxygen2', mode='nonbase')
[1] "brew"          "commonmark"    "desc"
[4] "digest"        "knitr"          "pkgload"
[7] "purrr"         "R6"             "Rcpp"
[10] "rlang"         "stringi"        "stringr"
[13] "xml2"          "assertthat"     "crayon"
[16] "rprojroot"     "evaluate"       "highr"
[19] "markdown"      "yaml"           "xfun"
[22] "cli"           "pkgbuild"       "rstudioapi"
[25] "withr"         "magrittr"       "glue"
[28] "fansi"         "mime"           "callr"
[31] "prettyunits"  "backports"      "processx"
[34] "ps"

> length(package.deps('roxygen2', mode='nonbase'))
[1] 34
```

# Package folders and files

```
[groth@bariuke build]$ ls -R pwr2020
```

```
pwr2020:
```

```
data  DESCRIPTION  LICENSE  man  NAMESPACE  R
```

```
pwr2020/data:
```

```
tdata.rda
```

```
pwr2020/man:
```

```
add2.Rd  add.Rd  pwr2020-package.Rd  tdata.Rd
```

```
pwr2020/R:
```

```
add2.R  add.R  pwr2020-internal.R
```

# Essentials

- Folder `R`: the R files with your code, multiple functions can be in the same file
- Folder `man`: The Rd manual files for your documentation, each function or dataset has its own file
- File `DESCRIPTION`: the meta information about your package (Name, Description, Author, Version, License, Dependencies (Imports, Depends))
- File `NAMESPACE`: the exported functions or data sets

# Optionals

- Folder `data`: R data sets which can be loaded using the `data` command
- Folder `inst`: additional files required by your package like images, Tcl scripts etc.
- Folder `vignettes`: tutorials on how to use your package.
- File `LICENSE`: if you provide a special license just as the MIT license
- File `NEWS` or `ChangeLog`: a file with information on updates in your package



# DESCRIPTION

```
> source('../scripts/file.head.R')  
> file.head('pwr2020/DESCRIPTION',10)
```

Package: pwr2020

Type: Package

Title: Package for course Programming with R

Version: 0.1

Date: 2020-09-18

Author: Detlef Groth, University of Potsdam

Maintainer: Who to complain to <dgroth@uni-potsdam.de>

Description: Collection of utility functions for the cour

License: MIT + file LICENSE

RoxygenNote: 6.1.1

# NAMESPACE

```
> file.head('pwr2020/NAMESPACE', 10)
exportPattern("^[:alpha:]+")
```

⇒ **See:** [https:](https://www.regular-expressions.info/posixbrackets.html)

[//www.regular-expressions.info/posixbrackets.html](https://www.regular-expressions.info/posixbrackets.html)

<code>[:alpha:]</code>	Alphabetic characters	<code>[a-zA-Z]</code>
# better export only lower case letters		
<code>[:lower:]</code>	lowercase letters	<code>[a-z]</code>

# R/add.R

```
> file.head('pwr2020/R/add.R', 10)
add <-
function (x, y)
{
    x + y
}
```

# man/add.Rd

```
> file.head('pwr2020/man/add.Rd', -1)
\name{add}
\title{
  add two numbers
}
\description{
  A intial starting function ...
}
\usage{
  add(x, y)
}
\arguments{
  \item{x}{
    numercial value
  }
  \item{y}{
```

```
    numerical value
  }
}
\details{
    Some more details ...
}
\value{
    return the sum of x and y
}
\author{
    Detlef Groth, University of Potsdam
}
\note{
    further notes ...
}

\seealso{
    See also % \code{\link{add2}}
}
```

```
\examples{  
  add(2, 3)  
  x=2  
  y=3  
  add(x, y)  
}
```

```
\keyword{ arith } % use one of RShowDoc("KEYWORDS")
```

# Minimal package: mini

Files:

- DESCRIPTION
- NAMESPACE
- LICENSE
- man/add.Rd
- R/add.R

⇒ That is a KISS approach !!

# Steps in Installation

- **build:** R CMD build pkgname
- **check:** R CMD check pkgname\_VERSION.tar.gz
- **install:** R CMD INSTALL pkgname\_VERSION.tar.gz



# Mini Build

```
#!/bin/bash
```

```
R CMD build mini |
```

```
perl -pe 's/[^-:\*\\/._A-Za-z0-9\s]+/"/g'
```

```
⇒ * checking for file "mini/DESCRIPTION" ... OK
```

```
⇒ * preparing "mini":
```

```
⇒ * checking DESCRIPTION meta-information ... OK
```

```
⇒ * checking for LF line-endings in source and make files
```

```
⇒ * checking for empty or unneeded directories
```

```
⇒ * building "mini_0.1.tar.gz"
```

 Rbuild.sh

⇒ the Perl onliner just for replacing special characters which LaTeX can't compile ...

# Mini Check

```
#!/bin/bash
```

```
R CMD check mini_0.1.tar.gz |
```

```
perl -pe 's/[^-\*\\/:._A-Za-z0-9\s]+/"/g' | tail
```

```
⇒ * checking Rd cross-references ... OK
```

```
⇒ * checking for missing documentation entries ... OK
```

```
⇒ * checking for code/documentation mismatches ... OK
```

```
⇒ * checking Rd "usage sections ... OK
```

```
⇒ * checking Rd contents ... OK
```

```
⇒ * checking for unstated dependencies in examples ... OK
```

```
⇒ * checking examples ... OK
```

```
⇒ * checking PDF version of manual ... OK
```

```
⇒ * DONE
```

```
⇒ Status: OK
```

 Rcheck.sh

# Mini INSTALL

```
#!/bin/bash
R CMD INSTALL --html mini_0.1.tar.gz \
  -l ../../../../myr/rlibs 2>&1 |
  perl -pe 's/[^-\*:\/.\_A-Za-z0-9\s]+/"/g'
⇒ * installing *source* package "mini" ...
⇒ ** using staged installation
⇒ ** R
⇒ ** byte-compile and prepare package for lazy loading
⇒ ** help
⇒ *** installing help indices
⇒   converting help for package "mini"
⇒     finding HTML links ... done
⇒     add                                                    html
⇒ ** building package indices
⇒ ** testing if installed package can be loaded from temp
⇒ ** testing if installed package can be loaded from fina
⇒ ** testing if installed package keeps a record of tempo
```

⇒ \* DONE "mini"



⇒ redirect stderr to stdout: 2>&1

# Result

```
[groth@bariuke]$ ls ../../myr/rlibs/mini
DESCRIPTION  help  html  INDEX  LICENSE
Meta  NAMESPACE  R
```

# Mini Test

```
> .libPaths(c(' ../../../../myr/', .libPaths()))  
> library(mini)  
> ls('package:mini')  
[1] "add"  
> mini::add(2, 3)  
[1] 5  
> add(2, 3)  
[1] 5  
> ?add
```

add                      package:mini                      R Documentation

add two numbers

Description:

A intial starting function ...

Usage:

add(x, y)

...

# Summary

- `install.packages`
- `library`
- `require`
- writing packages
- `devtools/roxygen2`
- minimal approach

## 4.4 Exercise 4 - packages

### Goals:

- we will create a minimal R-package called pwr2020
- we will learn how to build and install the package
- we will add 1-2 additional functions to the package such as `file.head(filename, n)` and `termcolor(color, txt)`
- On Windows we have to create batch files to aid in checking, building and installing.



# Task 1: Setup your working environment

- start Geany
- inside your `R-labs` directory creatr two other folder:
  - `rlibs` - therein will your packages installed
  - `rlibs-dev` - therein will your packages developed
- download the `mini.zip` file from Moodle (Download folder)
- unpack this `mini.zip` file into your `rlibs-dev` folder
- so you must have a folder `rlibs-dev/mini`
- Windows:
  - download the three Batch (`.bat`) files
  - place those three files into the `rlibs-dev` folder
  - so you should have a file like `rlibs-dev/Rcmdbuild.bat`

## Task 2: Building, Checking, Installing the mini package

- UNIX:
  - open a terminal and switch into the directory `R-labs/rlibs-dev`
  - run the command: `R CMD build mini`
  - a file `mini_0.1.tar.gz` should be generated
  - run the command `R CMD check -no-manual mini_0.1.tar.gz`
  - check if this runs ok
  - if this is the case run: `R CMD INSTALL -l ../rlibs -html mini_0.1.tar.gz`
- Windows:
  - Open a console/powershell and switch into the `R-labs/rlibs-dev` directory using the `cd` command
  - do a directory listing `ls` to check if your batch files are here

- run the command: `.\Rcmdbuild.bat mini`
- a file `mini_0.1.tar.gz` should be generated
- run the command `.\Rcmdcheck.bat`  
`mini_0.1.tar.gz`
- check if this runs ok
- if this is the case run: `.\Rcmdinstall.bat`  
`mini_0.1.tar.gz`

## Task 3: Test the mini package

- lets test first the mini package
- start R
- we have to extend the library path that R finds the mini package
- below is my R session on Windows which I used to test the package

```
> .libPaths(  
  c("C:/Users/Dr. Detlef Groth/Documents/Rlabs/rlibs"  
    .libPaths()))  
> library(mini)  
> mini::add(3, 4)  
7
```

⇒ Adapt the path and check if you can add two numbers mini package.  
⇒ Be sure to use the tab key for folder completion, don't write the long file name by Hand !!

## Task 4: Create a pwr2020 package

- pwr2020 because a pwr package exists already on CRAN
- Create a copy of the `mini` folder in the same directory and rename the new folder `pwr2020`.
- You should keep the `mini` package for future creations of new packages from scratch.
- So don't change anything in the `mini` package
- In Geany open the file browser sidebar left for easier navigation (Plugins->Filebrowser)
- Open the files `pwr2020/DESCRIPTION` and `pwr/LICENSE` in Geany and change the necessary details, author, package name etc.
- Create a new file `pwr2020/R/file_head.R` and add the code from the lecture to this file.
- Rename the function to `file_head`.
- Create a new file `pwr2020/man/file_head.Rd`
- Copy the manual from `add.Rd` to this file and adapt this to your

needs

- In the examples section add the following code:

```
\examples {  
file_head(file.path(system.file(package='pwr2020'),  
  "man", 'file_head.Rd'))  
}
```

- After adding the two files do again a package build, check and install.
- Test the new function in an interactive R session.

## Task 5: Add a **termcolor** function ...

to the package and use a few standard colors. Give a default color of green.

```
termcolor <- function (msg,col='GREEN') {  
  if (col=='GREEN') {  
    return(cat(codegreen,msg,resetcode))  
  } else if (col == 'RED') {
```

```
}  
  }  
  etc  
}
```

## Task 6 - Homework

- add the FastaUtils.R file from day 3 exercise to the package
- comment first only the read.fasta function
- do a stupid example  $x=1$  just for checking
- we will add a real fasta example later

# Programming with R

## Day 5

### R-GUIs

#### tcltk(2) package

## University of Potsdam

Detlef Groth



# Last Lecture

- `install.packages`
- `library`
- `require`
- writing packages
- `devtools/roxygen2`
- minimal approach

# Outline

## Day 1 (basics)

- setup, install editor, simple programs
- variables, operators
- data structures, control flow

## Day 2 (basics)

- functions
- file input/output
- terminal interaction
- command line arguments

## Day 3 (advanced)

- object oriented progr.
- code documentation
- R base graphics system

## Day 4 (advanced)

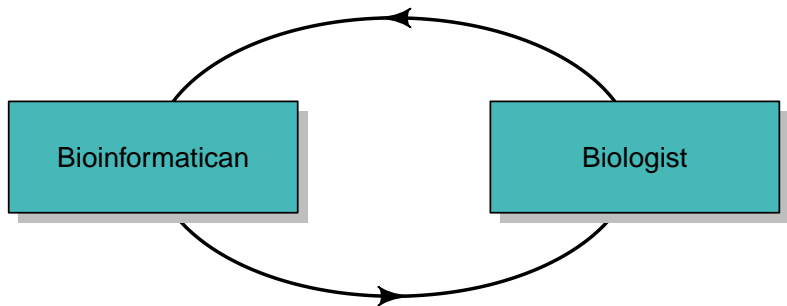
- using packages
- package documentation

## Day 5 (advanced)

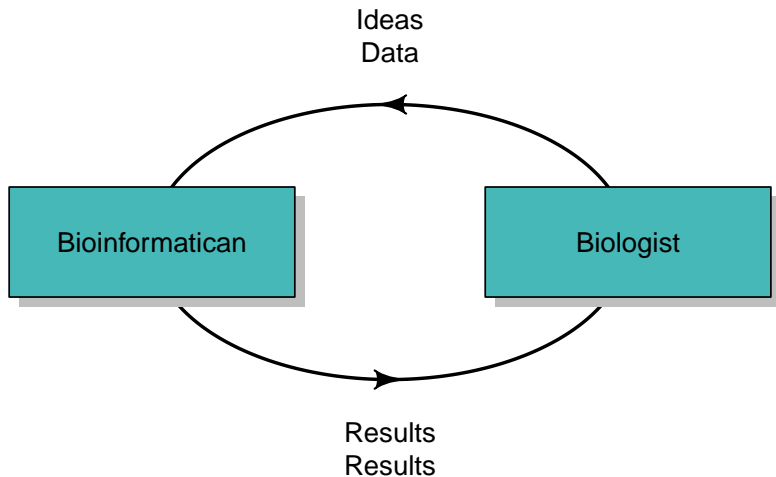
- **graphical user interfaces**
- **tcltk** (shiny)



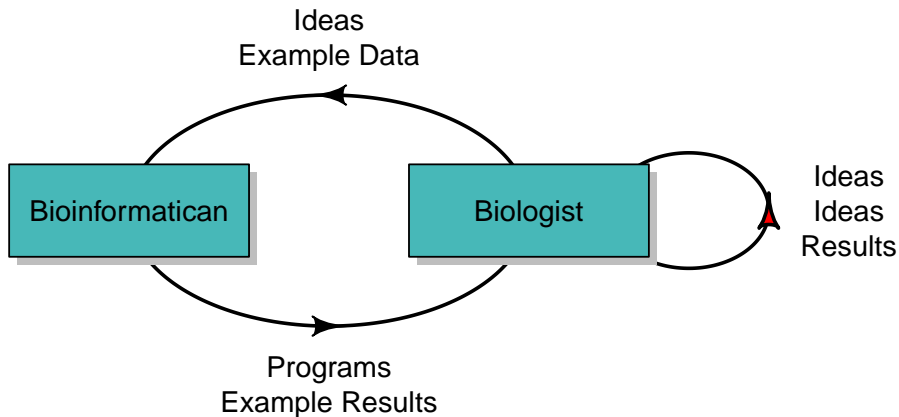
## 5 R-GUIs with tcltk



# Standard (?)



# The Better Way



# Principle

- almost no exchange of data
- almost only exchange of programs
- Pros:
  - you save time
  - you stick in the project
  - you can learn something new
- Cons:
  - you need to learn something new(?)

# Bioconductors Target Search Package

**BMC Bioinformatics**



Software

Open Access

## TargetSearch - a Bioconductor package for the efficient preprocessing of GC-MS metabolite profiling data

Álvaro Cuadros-Inostroza<sup>\*1,2</sup>, Camila Caldana<sup>1</sup>, Henning Redestig<sup>1,3</sup>, Miyako Kusano<sup>3</sup>, Jan Lisec<sup>1</sup>, Hugo Peña-Cortés<sup>2</sup>, Lothar Willmitzer<sup>1</sup> and Matthew A Hannah<sup>1,4</sup>

Address: <sup>1</sup>Max Planck Institute of Molecular Plant Physiology, Am Mühlenberg 1, D-14476 Potsdam-Golm, Germany, <sup>2</sup>Centro de Biotecnología, Universidad Técnica Federico Santa María, General Bari 699 Valparaíso, Chile, <sup>3</sup>RIKEN Plant Science Center, Tsurumi-ku, Suehiro-cho, 1-7-22 Yokohama, Kanagawa, 230-0045, Japan and <sup>4</sup>Bayer BioScience N.V., Technologiepark 38, B-9052, Gent, Belgium

Email: Álvaro Cuadros-Inostroza<sup>\*</sup> - [inoostroza@mpimp-golm.mpg.de](mailto:inoostroza@mpimp-golm.mpg.de); Camila Caldana - [caldana@mpimp-golm.mpg.de](mailto:caldana@mpimp-golm.mpg.de); Henning Redestig - [henning.red@psc.riken.jp](mailto:henning.red@psc.riken.jp); Miyako Kusano - [mkusano005@psc.riken.jp](mailto:mkusano005@psc.riken.jp); Jan Lisec - [lisec@mpimp-golm.mpg.de](mailto:lisec@mpimp-golm.mpg.de); Hugo Peña-Cortés - [hugo.pena@usm.cl](mailto:hugo.pena@usm.cl); Lothar Willmitzer - [willmitzer@mpimp-golm.mpg.de](mailto:willmitzer@mpimp-golm.mpg.de); Matthew A Hannah - [matthew.hannah@bayercropsience.com](mailto:matthew.hannah@bayercropsience.com)

<sup>\*</sup> Corresponding author

Published: 16 December 2009

Received: 20 August 2009

BMC Bioinformatics 2009, 10:428 doi:10.1186/1471-2105-10-428

Accepted: 16 December 2009

This article is available from: <http://www.biomedcentral.com/1471-2105/10/428>

© 2009 Cuadros-Inostroza et al; licensee BioMed Central Ltd.

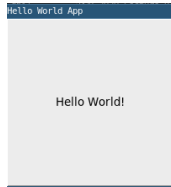
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/2.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



# GUI Aims

- providing an easy to use interface for the user to interact with an application
- CLI: commandline interfaces reduced interaction, good for automating, worser for intuitive usage
- GUI: graphical user interface good for interactive usage, bad for automated usage

# 5.1 Hello World



```
> library(tcltk) # library
> tt=tkoplevel() # main window
> tkwm.title(tt,"Hello World App") # app name

<Tcl>

> tkpack( # geometry manager
+     ttkbutton( # widget
+     tt,text="Hello World!", # parent and options
+     command=function () { # callback
+         tkdestroy(tt)
+     } ),
+     fill='both',expand=TRUE) # geometry options

<Tcl>

> tkdestroy(tt) # just to close it in my script
```

# Hello World in steps

```
> library(tcltk2) # more widgets herein
> hw = function () { print('Hello World'); }
> tt=tktoplevel()
> tkwm.title(tt, 'DGApp')

<Tcl>
> tkbtn=ttkbutton(tt,text='Hello World',
+      command=hw)
> tkpack(tkbtn,side='top',expand=TRUE,
+      fill='both')

<Tcl>
> tklbl=ttklabel(tt,text='LabelText')
> tkpack(tklbl)

<Tcl>
> tkdestroy(tt)
```

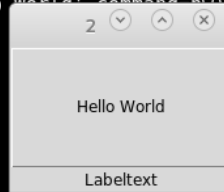
```

tt=tktoplevel()
+ tkbutton(tt,text='Hello World',command=hw)
ID
1] ".2.1"

env
environment: 0x16f48c8>

+ ttr(",class")
1] "tkwin"
+ tkpack(tkbtn,side='top',expand=TRUE,fill='both')
error in .Tcl.args.objv(...) : object 'tkbtn' not found
+ tkbtn=tkbutton(tt,text='Hello World',command=hw)
+ tkpack(tkbtn,side='top',expand=TRUE,fill='both')
Tcl>
+ tklbl=tklabel(tt,text='Labeltext')
+ tkpack(tkLBL)
Tcl>
[1] "Hello World"

```



# Hello World observations

- tktoplevel - main window
- tkwm - properties of main window
- tkpack - geometry manager
- ttkbutton, ttklabel - widget
- command option - callback with function

# Tk: R vs Python3

- Python:

```
1 root = tk.Toplevel()  
2 button = tk.Button(root)  
3 button.pack()  
- object.method()
```

- R:

```
1 root = tktoplevel()  
2 button = tkbutton(root)  
3 tkpack(button)  
- tkmethod(object)
```

⇒ **One API for Python, Perl, R, Ruby, C++ and Tcl/Tk!!**

# Advantages of Tcl/Tk API

- installed with every R (Mac OSX?)
- crossplatform (all major OS)
- lightweight
- cross language (Perl, Python, Ruby, R, ...)
- can be easily extended
- highly introspective as R

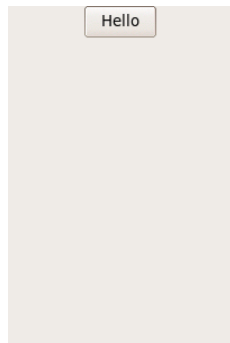
## 5.2 Layout Widgets

### Toplevel

- toplevel is a window of the application
- you can have several toplevel in the same application
- toplevels contain other widgets, like frames, labels, buttons, checkboxes, menus etc.
- every GUI application needs a toplevel
- you can set the title of a toplevel and other properties like width, height and position on the screen



```
> tt=tktoplevel()  
> tkwm.title(tt, 'DGApp')  
<Tcl>  
> tkwm.geometry(tt, '140x200')  
<Tcl>  
> tkpack(ttkbutton(tt, text='Hello'))  
<Tcl>  
> source('../scripts/screenshot.r')  
> screenshot(tt, 'toplevel.png')  
> tkdestroy(tt)
```



—  screenshot.r

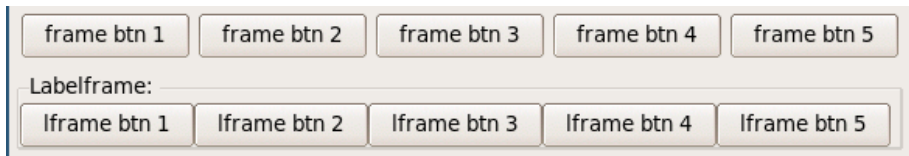
# What other tkwm commands are available?

```
> options(width=55)
> grep("tkwm",ls("package:tcltk"),value=TRUE)
[1] "tkwm.aspect"          "tkwm.client"
[3] "tkwm.colormapwindows" "tkwm.command"
[5] "tkwm.deiconify"       "tkwm.focusmodel"
[7] "tkwm.frame"           "tkwm.geometry"
[9] "tkwm.grid"            "tkwm.group"
[11] "tkwm.iconbitmap"      "tkwm.iconify"
[13] "tkwm.iconmask"        "tkwm.iconname"
[15] "tkwm.iconposition"    "tkwm.iconwindow"
[17] "tkwm.maxsize"         "tkwm.minsize"
[19] "tkwm.overrideredirect" "tkwm.positionfrom"
[21] "tkwm.protocol"        "tkwm.resizable"
[23] "tkwm.sizefrom"         "tkwm.state"
[25] "tkwm.title"           "tkwm.transient"
[27] "tkwm.withdraw"
```

# Frames / Labelframes

```
> tt=tktoplevel()
> tkframe=tkkframe(tt)
> for (i in 1:5) {
+   tkpack(ttkbutton(tkframe,
+   text=paste('frame btn',i)),side='left',
+   padx=3)
+ }
> tkiframe=ttklabelframe(tt,text='Labelframe: ')
> for (i in 1:5) {
+   tkpack(ttkbutton(tkiframe,
+   text=paste('lframe btn',i)),side='left')
+ }
> tkpack(tkframe, padx=5, pady=5)
<Tcl>
> tkpack(tkiframe, padx=5, pady=5, fill='both',
+   expand=TRUE)
<Tcl>
```

```
> screenshot(tt, 'toplevel-02.png')  
> tkdestroy(tt)
```



# Layout manager commands

- **tkpack**, just pack into the current container
- **options** <http://www.tcl.tk/man/tcl/TkCmd/pack.htm>
  - padx, pady amount of space around the widget
  - ipadx, ipady internal padding
  - expand: should container propagate if parent widget size is changed
  - fill: x,y or both should container fill space in x and/or y direction
- **tkgrid**, matrix like layout:
- **options** <http://www.tcl.tk/man/tcl/TkCmd/grid.htm>
  - column, columnspan, row, rowspan
  - padx, pady, ipadx, ipady

## 5.3 Basic Widgets

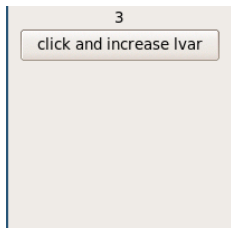
- Frames, layout container
- Labels (tklabel, ttklabel)
- Buttons (tkbutton, ttkbutton)
- Checkbutons (tkcheckboxbutton, ttkcheckboxbutton)
- Radiobutons (tkradiobutton,ttkradiobutton)
- Entries (tkentry, ttkentry)
- Comboboxes (ttkcombobox)

# Label and Button

```
> tt=tktoplevel()
> lvar=tclVar('1')
> increaseLvar = function () {
+     tclvalue(lvar) =
+     as.integer(tclvalue(lvar)) +1
+ }
> tk1=ttklabel(tt,textvariable=lvar)
> tkbtn=ttkbutton(tt,command=increaseLvar,
+     text='click and increase lvar')
> tkpack(tk1)
<Tcl>
> tkpack(tkbtn)
<Tcl>
> tkinvoke(tkbtn)
<Tcl> 2
> tkinvoke(tkbtn)
```

```
<Tcl> 3
```

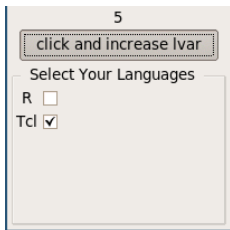
```
> screenshot(tt, 'textvar.png')
```





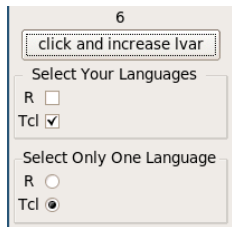
# Checkbutton, Radiobuttons

```
> tklf=tklabelframe(tt,  
+      text='  Select Your Languages  ')  
> checkedR=tclVar('0')  
> checkedTcl=tclVar('1')  
> tkgrid(tklabel(tklf,text='R'),  
+      ttkcheckbutton(tklf,variable=checkedR))  
<Tcl>  
> tkgrid(tklabel(tklf,text='Tcl'),  
+      ttkcheckbutton(tklf,variable=checkedTcl))  
<Tcl>  
> tkpack(tklf,padx=4,pady=4,fill='both',  
+      expand=TRUE)  
<Tcl>  
> screenshot(tt,'checkbutton.png')
```



```
> tklf=tktklabelframe(tt,  
+     text='Select Only One Language ')  
> checkedLang=tclVar('Tcl')  
> tkgrid(ttklabel(tklf,text='R'),  
+        ttkradiobutton(tklf,variable=checkedLang,  
+                        value='R'))  
<Tcl>  
> tkgrid(ttklabel(tklf,text='Tcl'),  
+        ttkradiobutton(tklf,variable=checkedLang,  
+                        value='Tcl'))  
<Tcl>
```

```
> tkpack (tkl f, padx=4, pady=4, fill='both',  
+         expand=TRUE)  
<Tcl>  
> screenshot (tt, 'radiobutton.png')  
> tkdestroy (tt)
```



# Entry

```
> tt=tktoplevel()
> tkwm.geometry(tt, '300x150')
<Tcl>
> tklf=ttklabelframe(tt, text='rnorm settings')
> rn1Mean=tclVar('1')
> rn1Sd=tclVar('2')
> rn2Mean=tclVar('1')
> rn2Sd=tclVar('1')
> asInt = function (vname) {
+   return(as.integer(tclvalue(vname)))
+ }
> plotXY = function () {
+   pdf('test-plot.pdf')
+   rn1=rnorm(100, mean=asInt(rn1Mean),
+   sd=asInt(rn1Sd))
+   rn2=rnorm(100, mean=asInt(rn2Mean),
```

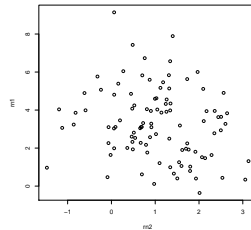
```
+      sd=asInt(rn2Sd))
+      print(head(rn1))
+      plot(rn1~rn2)
+      dev.off()
+ }
> tkgrid(tklabel(tklf,text=''),ttklabel(tklf,
+   text='Mean'),ttklabel(tklf,text='SD'))
<Tcl>
> tkgrid(ttklabel(tklf,text='RN1'),
+   ttkentry(tklf,textvariable=rn1Mean,width=10),
+   ttkentry(tklf,textvariable=rn1Sd,width=10))
<Tcl>
> tkgrid(ttklabel(tklf,text='RN2'),
+   ttkentry(tklf,textvariable=rn2Mean,width=10),
+   ttkentry(tklf,textvariable=rn2Sd,width=10))
<Tcl>
> cmdBtn=ttkbutton(tt,command=plotXY,
+   text='Los plotte!!')
> tkpack(cmdBtn)
```

```
<Tcl>  
> tkpack(tk1f, padx=5, pady=5)  
<Tcl>  
> tclvalue(rn1Mean)='3'  
> tkinvoke(cmdBtn)  
[1] 0.3986878 1.9010668 4.3225695 1.6391491 4.6223129  
[6] 3.4579258  
<Tcl>  
> screenshot(tt, 'plotxy.png')
```

Los plotte!!

rnorm settings

	Mean	SD
RN1	3	2
RN2	1	1



## 5.4 Documentation

```
> options(useFancyQuotes = FALSE)
> getAnywhere('tkwm.geometry')
```

A single object matching 'tkwm.geometry' was found  
It was found in the following places

```
package:tcltk
namespace:tcltk
with value
```

```
function (...)
tcl("wm", "geometry", ...)
<bytecode: 0x562f816b1378>
<environment: namespace:tcltk>
```

Try: `$ man n wm` and search for `/wm geometry`

```
> getAnywhere('tkdestroy')
```



A single object matching 'tkdestroy' was found  
It was found in the following places

```
package:tcltk  
namespace:tcltk  
with value
```

```
function (win)  
{  
  tcl("destroy", win)  
  ID <- .Tk.ID(win)  
  env <- get("parent", envir = win$env)$env  
  if (exists(ID, envir = env, inherits = FALSE))  
    rm(list = ID, envir = env)  
}  
<bytecode: 0x562f818819d8>  
<environment: namespace:tcltk>
```

```
> getAnywhere('ttknotebook')
```

A single object matching 'ttknotebook' was found  
It was found in the following places  
package:tcltk  
namespace:tcltk  
with value

```
function (parent, ...)  
tkwidget(parent, "ttk::notebook", ...)  
<bytecode: 0x562f80b04df8>  
<environment: namespace:tcltk>
```

Try: `$ man n ttk::notebook`

```
ttk::notebook(n)                                Tk Themed Widget                                ttk::notebook(n)
```

---

#### NAME

`ttk::notebook` - Multi-paned container widget

#### SYNOPSIS

```
ttk::notebook pathname ?options...?  
pathname add window ?options...?  
pathname insert index window ?options...?
```

---

#### DESCRIPTION

A `ttk::notebook` widget manages a collection of windows and displays a single one at a time. Each slave window is associated with a tab, which the user may select to change the currently-displayed window.

# R vs Tcl/Tk

```
# Session in R
tt <- tktoplevel()
b <- ttkbutton(tt,
  text="Click Me")
tkpack(b, side = "top",
  anchor="w",
  padx=3, pady=3)
foo <- function() {
  print("Hello World")
}
tkconfigure(b,
  command=foo)
tkcget(b, text=NULL)
```

```
# Same Session in Tcl
set tt [toplevel .1]
set b [ttk::button $tt.b \
  -text "Click Me"]
pack $b -side top -anchor w \
  -padx 3 -pady 3
proc foo {} {
  puts "Hello World"
}
$b configure -command foo
$b cget -text
```



## 5.5 Dialogs

- tkmessageBox
- tkgetOpenFile
- tkgetSaveFile
- tkchooseDirectory
- (tkchooseColor)
- (tkchooseFont)

# tkmessageBox

```
> #screenshot('Greetings','messagebox-01.png')
> res <- tkmessageBox(title = "Greetings",
+   message = "Hello, world!", icon = "info",
+   type = "ok")
> res           # This is a Tcl variable
<Tcl> ok
> ## <Tcl> ok
> tclvalue(res) # Get the value from a Tcl variable
[1] "ok"
> ## [1] "ok"
> as.character(res) # It works also that way
[1] "ok"
> ## [1] "ok"
```

Other types: [http:](http://www.tcl.tk/man/tcl/TkCmd/messageBox.htm)

[//www.tcl.tk/man/tcl/TkCmd/messageBox.htm](http://www.tcl.tk/man/tcl/TkCmd/messageBox.htm)

```
tkmessageBox(  
    message = "An error has occurred!",  
    icon = "error", type = "ok")  
tkmessageBox(  
    message = "Do you want to save before quitting?",  
    icon = "question", type = "yesnocancel",  
    default = "yes")
```

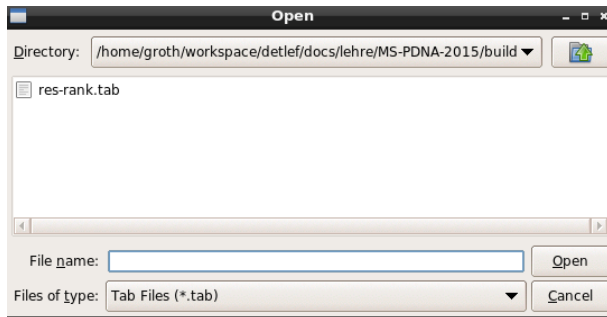


# tkgetOpenFile

```
> getTabFileData <- function() {  
+   name <- tclvalue(tkgetOpenFile(title = 'Open',  
+     filetypes = "{{Tab Files} {.tab}}  
+     {{CSV Files} {.csv}}  
+     {{All Files} *}"))  
+   if (name == "") { #Return an empty data frame  
+     return(data.frame())#if no file was selected  
+   }  
+   data <- read.table(name, sep="\t", header=TRUE)  
+   assign("tab_data", data, envir = .GlobalEnv)  
+ }  
> tt=tktoplevel()  
> btn=ttkbutton(tt, text="Load Tab Data",  
+   command=getTabFileData)  
> tkpack(btn)  
<Tcl>
```



```
> tkinvoke(btn)
<Tcl>
> tkdestroy(tt)
```



# tkgetSaveFile

- Example: save the current plot to a png

```
> png_filename <- tclvalue(tkgetSaveFile(  
+   title='Save', initialfile = "test.png",  
+   filetypes = "{{PNG Files} {.png}}  
+   {{JPG Files} {.jpg .jpeg}} {{All Files} * }"))  
> print(png_filename)  
[1] ""
```

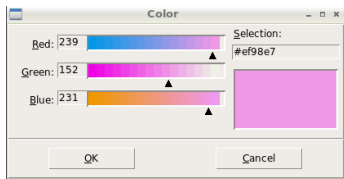
# tkchooseDirectory

```
> opendir<-function(...) {  
+   return(tclvalue(tkchooseDirectory(...)))  
+ }  
  
> openfile<-function(...) {  
+   return(tclvalue(tkgetOpenFile(...)))  
+ }  
  
> openfile(title='Open any file ...')  
[1] "/home/groth/workspace/delfgroth/docs/lehre/PwR-2020/
```

# tkchooseColor

- Not implemented in R-tcltk but in Tcl/Tk itself.
- Let's implement it:

```
> tkchooseColor=function(...)  
+   tcl("tk_chooseColor", ...)  
> col=tkchooseColor(title='Color')  
> col  
<Tcl> #efb4e7  
> tclvalue(col)  
[1] "#efb4e7"
```

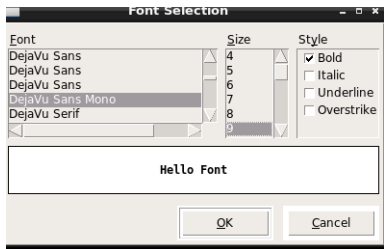


# tkchooseFont

- Not in R tcltk, tcltk2
- But in standard TclTk

⇒ our own implementation

```
> tkchooseFont = function (...) {  
+   tcl('tk', 'fontchooser', 'configure', ...)  
+   tcl('tk', 'fontchooser', 'show')  
+ }  
> tt=tktoplevel()  
> font=tkchooseFont(title='Font Selection',  
+   parent=tt)  
> print(tclvalue(font))  
[1] " "
```

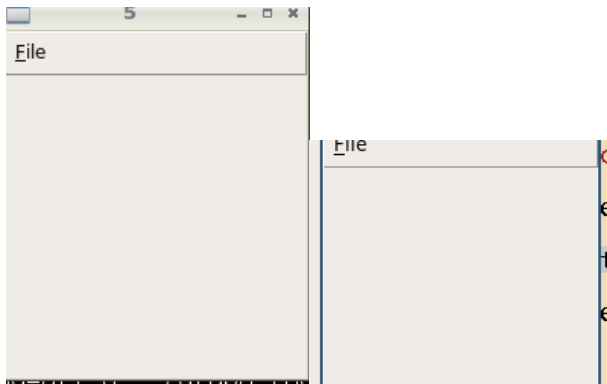


## 5.6 Menues

```
> openfile=function() {}  
> opendir=function() {}  
> tt<-tktoplevel()  
> topMenu<-tkmenu(tt,tearoff=FALSE)  
> tkconfigure(tt,menu=topMenu)  
<Tcl>  
> fileMenu<-tkmenu(topMenu, tearoff=FALSE)  
> tkadd(fileMenu,"command",label="Open file",  
+       command=openfile,underline=0)  
<Tcl>  
> tkadd(fileMenu, "command",  
+       label="Open directory",  
+       command=opendir,underline=5)  
<Tcl>  
> tkadd(fileMenu,'separator')  
<Tcl>
```

```
> tkadd(fileMenu, "command", label="Quit",
+       command=function() tkdestroy(tt),
+       underline=0)
<Tcl>
> tkadd(topMenu, "cascade", label="File",
+       menu=fileMenu,
+       underline=0)
<Tcl>
> tkfocus(tt)
<Tcl>
> # just for screenshot
> geo=as.character(tkwininfo('geometry',tt))
> geo=strsplit(geo,"\\+",perl=TRUE)[[1]]
> tkpost(fileMenu,as.integer(geo[2])+2,
+       as.integer(geo[3])+8)
<Tcl>
> screenshot(tt, 'tcltk-menu-post.png', rootcrop=TRUE)
> tkdestroy(tt)
```





## 5.7 Advanced Widgets

### Combobox

A Entry widget with a dropdown list. Content can be tied as well to a tcl variable

```
> tt=tktoplevel()
> fruit=tclVar('Orange')
> tkcombo=ttkcombobox(tt,textvariable=fruit)
> tkconfigure(tkcombo,
+     values=c('Apple','Orange','Banana'))
<Tcl>
> tkpack(tkcombo)
<Tcl>
> #screenshot(tt,'ttkcombo.png')
> tkdestroy(tt)
>
```



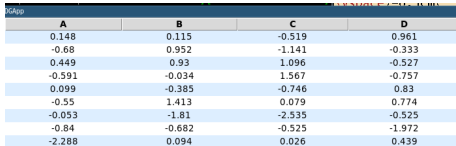
# Table (Treeview)-Widget

```
> tt=tktoplevel()
> tkwm.title(tt, 'DGApp')
<Tcl>
> tview=ttktreeview(tt, columns=LETTERS[1:4],
+      show='headings')
> # our own implementation
> tkheading = function (widget,...) {
+     tcl(widget, 'heading', ...)
+ }
> for (l in LETTERS[1:4]) {
+     tkheading(tview, l, text=l)
+     tcl(tview, "column", l, anchor='center')
+ }
> tktag.configure(tview, "band0", background=' #FFFFFF')
<Tcl>
> tktag.configure(tview, "band1", background=' #DDEEFF')
```

```

<Tcl>
> for (i in 1:30) {
+     band=paste('band',i%2,sep='')
+     item=tkinsert(tview,'','end',
+         values=round(rnorm(4),3))
+     tktag.add(tview,band,item)
+ }
> tkpack(tview,side='top',fill='both',expand=TRUE)
<Tcl>
> #screenshot(tt,'tcltk-04.png')

```



A	B	C	D
0.148	0.115	-0.519	0.961
-0.68	0.952	-1.141	-0.333
0.449	0.93	1.096	-0.527
-0.591	-0.034	1.567	-0.757
0.099	-0.385	-0.746	0.83
-0.55	1.413	0.079	0.774
-0.053	-1.81	-2.535	-0.525
-0.84	-0.682	-0.525	-1.972
-2.288	0.094	0.026	0.439

## Manualpage:

[https://www.tcl.tk/man/tcl8.6/TkCmd/ttk\\_treeview.htm](https://www.tcl.tk/man/tcl8.6/TkCmd/ttk_treeview.htm)

# Scrollbars

```
> tkscrolled = function (parent,widget,...) {  
+   scrx <- tk2scrollbar(parent,  
+       orient = 'horizontal',  
+       command = function(...) tkxview(widget, ...))  
+   scry <- tk2scrollbar(parent, orient = 'vertical',  
+       command = function(...) tkyview(widget, ...))  
+   tkconfigure(widget, xscrollcommand =  
+       function(...) tkset(scrx, ...),  
+       yscrollcommand = function(...) tkset(scry, ...))  
+   tkgrid(widget, scry, sticky = 'nsew')  
+   tkgrid.rowconfigure(parent, widget, weight = 1)  
+   tkgrid.columnconfigure(parent, widget, weight = 1)  
+   tkgrid(scrx, sticky = 'ew')  
+ }  
> tkpack.forget(tvview)  
<Tcl>  
> tkscrolled(tt,tview)
```

```
<Tcl>
```

```
> #screenshot(tt, 'tcltk-06.png')
```

```
> tkdestroy(tt)
```



The screenshot shows a Tkinter window titled "DGApp" with a table widget. The table has four columns labeled A, B, C, and D. The data is as follows:

A	B	C	D
0.148	0.115	-0.519	0.961
-0.68	0.952	-1.141	-0.333
0.449	0.93	1.096	-0.527
-0.591	-0.034	1.567	-0.757
0.099	-0.385	-0.746	0.83
-0.55	1.413	0.079	0.774
-0.053	-1.81	-2.535	-0.525
-0.84	-0.682	-0.525	-1.972

# Autoscroll - Implementation

```
> tk2autoscroll = function (parent,widget,...) {  
+   tclRequire('autoscroll')  
+   scrx <- tk2scrollbar(parent, orient = 'horizontal',  
+       command = function(...) tkxview(widget, ...))  
+   scry <- tk2scrollbar(parent,orient = 'vertical',  
+       command = function(...) tkyview(widget, ...))  
+   tkconfigure(widget, xscrollcommand =  
+       function(...) tkset(scrx, ...),  
+       yscrollcommand = function(...) tkset(scry, ...))  
+   tkpack(scrx,side='bottom',fill='x')  
+   tkpack(scry,side='right',fill='y')  
+   tkpack(widget,side='top',fill='both',expand=TRUE)  
+   tcl('::autoscroll::autoscroll',scry)  
+   tcl('::autoscroll::autoscroll',scrx)  
+ }
```



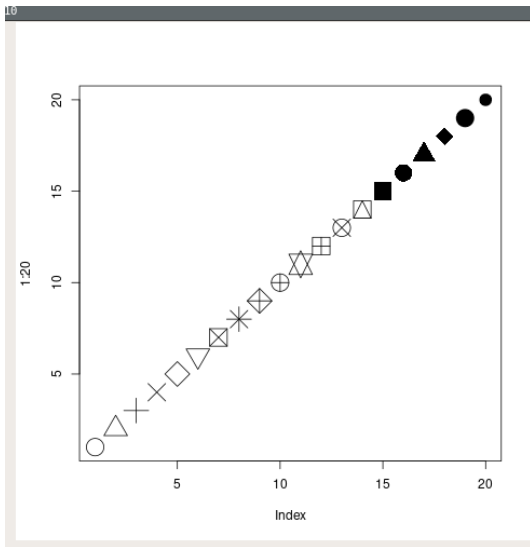
# Autoscroll with Text widget

```
> tt=tktoplevel()
> txt=tktext(tt,wrap='none')
> tk2autoscrollled(tt,txt)
<Tcl> .13.2
> for (i in 1:30) {
+     for (j in 1:30) {
+         tkinsert(txt,'end',j)
+     }
+     tkinsert(txt,'end',"\\n")
+ }
> tkwm.geometry(tt,'300x400+30+30')
<Tcl>
> screenshot(tt,'tcltk-autoscroll-01.png',update=TRUE)
> tkwm.geometry(tt,'500x600+30+30')
<Tcl>
> #screenshot(tt,'tcltk-autoscroll-02.png',update=TRUE)
> try(tkdestroy(tt))
```



# Images

```
> tt=tktoplevel()
> png('test-plot-01.png')
> plot(1:20,pch=1:20,cex=3)
> dev.off()
null device
      1
> imgfile = 'test-plot-01.png'
> image1 = tclVar()
> tkimage.create('photo', image1, file = imgfile)
<Tcl> ::RTcl10
> tkpack(tklabel(tt,image=image1))
<Tcl>
> tkdestroy(tt)
```



## 5.8 Advanced Layout Widgets

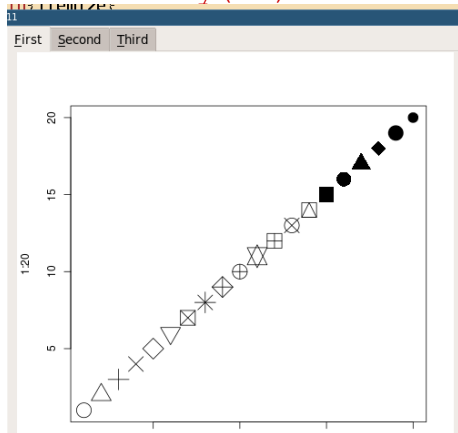
### Layout - Notebooks

- container widget
- `ttk::notebook`
- `http://www.tcl.tk/man/tcl/TkCmd/ttk\_notebook.htm`
- no need to pack added pages
- tabs are added using `tkadd`

```
> tt=tktoplevel()
> tknb=ttknotebook(tt)
> tf1=ttkframe(tknb)
> tf2=ttkframe(tknb)
> tf3=ttkframe(tknb)
> tkadd(tknb,tf1,text='First',underline=0)
```

```
<Tcl>
> tkadd(tknb,tf2,text='Second',underline=0)
<Tcl>
> tkadd(tknb,tf3,text='Third',underline=0)
<Tcl>
> tcl('::ttk::notebook::enableTraversal', tknb)
<Tcl> .15.1
> args(tk2notetraverse)
function (nb)
NULL
> body(tk2notetraverse)
{
    res <- tcl("ttk::notebook::enableTraversal", nb)
    return(invisible(res))
}
> tkpack(tknb,side='top',expand=TRUE,fill='both')
<Tcl>
```

```
> tkpack(tklabel(tf1, image=image1))  
<Tcl>  
> tkpack(tkbutton(tf1, text='Hello'))  
<Tcl>  
> tkdestroy(tt)
```



# Layout - Panedwindow

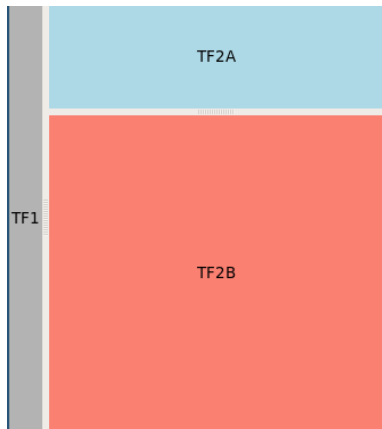
- container widget
- added widgets can be resized by dragging at the sash of the panedwindow

```
> tt=tktoplevel()  
> tfpw=ttkpanedwindow(tt,orient='horizontal',  
+      width=600,height=400)  
> tf1=ttkframe(tfpw)  
> tf2=ttkframe(tfpw)  
> tf2pw=ttkpanedwindow(tf2,orient='vertical',  
+      height=400,width=400)  
> tf2a=ttkframe(tf2pw)  
> tf2b=ttkframe(tf2pw)  
> tkadd(tfpw,tf1)  
<Tcl>  
> tkadd(tfpw,tf2)
```



```
<Tcl>
> tkadd(tf2pw,tf2a)
<Tcl>
> tkadd(tf2pw,tf2b)
<Tcl>
> tkpack(tf2pw,side='top',expand=TRUE,
+       fill='both')
<Tcl>
> # ttklabel has no bg option
> tkpack(ttklabel(tf1,text='TF1',bg='grey70'),
+       expand=TRUE,fill='both')
<Tcl>
> tkpack(ttklabel(tf2a,text='TF2A',bg='light blue'),
+       expand=TRUE,fill='both')
<Tcl>
> tkpack(ttklabel(tf2b,text='TF2B',bg='salmon'),
+       expand=TRUE,fill='both')
<Tcl>
```

```
> # last the main widget at best to avoid flicker  
> tkpack(tfpw, side='top', expand=TRUE, fill='both')  
<Tcl>  
> tkdestroy(tt)
```



## 5.9 Events

Often specific user actions needs to be captured.

Summary: <http://www.sciviews.org/recipes/tcltk/TclTk-event-binding/>

- keyboard events <Keypress-Ctrl-a>
- mouse events <Mouse-1> left mouse button click
- virtual events like «Tabchanged», <Enter> (mouse pointer enters a widget)

Syntax:

```
tkbind(widget, 'event', function)
```

# Examples

```
> tt=tktoplevel()
> entering = function () {
+     print('Entering the Label ...')
+ }
> leaving = function () {
+     print('leaving the label')
+ }
> tk1=ttklabel(tt,text='enter me')
> tkbind(tk1, '<Enter>',entering)
<Tcl>
> tkbind(tk1, '<Leave>',leaving)
<Tcl>
> tkpack(tk1)
<Tcl>
> tkdestroy(tt)
```

```
> source("../test.r")
> [1] "Entering the Label ..."
[1] "leaving the label"
[1] "Entering the Label ..."
[1] "leaving the label"
[1] "Entering the Label ..."
[1] "leaving the label"
[1] "Entering the Label ..."
[1] "leaving the label"
```




# Virtual Events

```
> tt=tktoplevel()
> tn=ttknotebook(tt)
> # wrapper function
> # ttk::notebook has a tab method which
> # is not wrapped by tcltk2
> tktab = function (widget,...) {
+   tcl(widget, 'tab', ...)
+ }
> # event function
> changeTab = function () {
+   current=tclvalue(tkindex(tn, 'current'))
+   # can't use text ... Tcl-like option -text
+   print(paste("Tab",
+       tktab(tn,current, '-text'), "is active"))
+   # R like option text=NULL
+   print(paste("Tab",
```

```
+ tktab(tn,current,text=NULL),"is active"))
+
+ }
> tkbind(tn,"<<NotebookTabChanged>>",changeTab)
<Tcl>
> ttkf1=ttkframe(tn)
> ttkf2=ttkframe(tn)
> tkadd(tn,ttkf1,text="First",underline=0)
<Tcl>
> tkadd(tn,ttkf2,text="Second",underline=0)
<Tcl>
> tkpack(tn)
<Tcl>
> tkdestroy(tt)
```

```
> source("../test.r")
[1] "Tab First is active"
> [1] "Tab Second is active"
[1] "Tab First is active"
[1] "Tab Second is active"
```



# Binding Parameters

Sometimes we would like to know which widgets has gotten an event.

<http://www.tcl.tk/man/tcl/TkCmd/bind.htm>  
(Substitutions)

- important substitutions:
  - %x => xclick coordinates
  - %y => yclick coordinates
  - %W => which widget got the event
  - many others ...

The R syntax is slightly different:

```
> .Tcl.callback(function(W,x,y) cat(W,x,y, '\n'))  
[1] "R_call 0x562f8099afc0 %W %x %y"
```




# Binding Parameters - Notebook Example

```
> tt=tktoplevel()
> tn=ttknotebook(tt)
> # wrapper function
> # ttk::notebook has a tab method which
> # is not wrapped by tcltk2
> tktab = function (widget,...) {
+     tcl(widget,'tab', ...)
+ }
> # event function
> changeTab = function (widget) {
+     # no global variable anymore
+     current=tclvalue(tkindex(widget,'current'))
+     # can't use text ...
+     print(paste("Tab",
+         tktab(widget,current,'-text'), "is active"))
+ }
```

```
> tkbind(tn, "<<NotebookTabChanged>>",  
+ function(W) changeTab(W) )  
<Tcl>  
> ttkf1=ttkframe(tn)  
> ttkf2=ttkframe(tn)  
> tkadd(tn,ttkf1,text="First",underline=0)  
<Tcl>  
> tkadd(tn,ttkf2,text="Second",underline=0)  
<Tcl>  
> tkpack(tn)  
<Tcl>  
> tkdestroy(tt)
```

```
source("../test.R")  
[1] "Tab First is active"  
[1] "Tab First is active"  
> [1] "Tab Second is active"  
[1] "Tab First is active"  
[1] "Tab Second is active"  
[1] "Tab First is active"
```



# Binding Parameters - XY - Example

```
> tt=tktoplevel()
> clickLabel = function (W,x=0,y=0) {
+   # Tcl like with minus
+   print(paste("Label ",tkcget(W, '-text'),
+     "clicked!"))
+   # R like with NULL
+   print(paste("Label ",tkcget(W, text=NULL),
+     "clicked!"))
+   if(x>0) {
+     print(paste("x =", x))
+     print(paste("y =", y))
+   }
+ }
> tk12=tklabel(tt,text='Click-me!')
> tkbind(tk12, '<Button-1>',
+   function(W) { clickLabel(W) })
```

<Tcl>

```
> tk13=tklabel(tt,text='Click-meXY!')
```

```
> tkbind(tk13, '<Button-1',
```

```
+ function(W,x,y){ clickLabel(W,x,y) })
```

<Tcl>

```
> tkpack(tk12)
```

<Tcl>

```
> tkpack(tk13)
```

<Tcl>

```
> tkdestroy(tt)
```

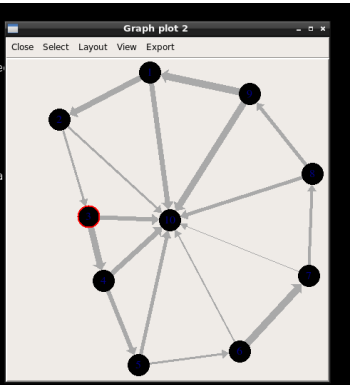
```
> source("../test.tcl")
> [1] "Label Click-meXY! clicked!"
[1] "x = 43"
[1] "y = 8"
[1] "Label Click-meXY! clicked!"
[1] "x = 43"
[1] "y = 8"
[1] "Label Click-meXY! clicked!"
[1] "x = 39"
[1] "y = 8"
[1] "Label Click-meXY! clicked!"
[1] "x = 39"
[1] "y = 8"
```



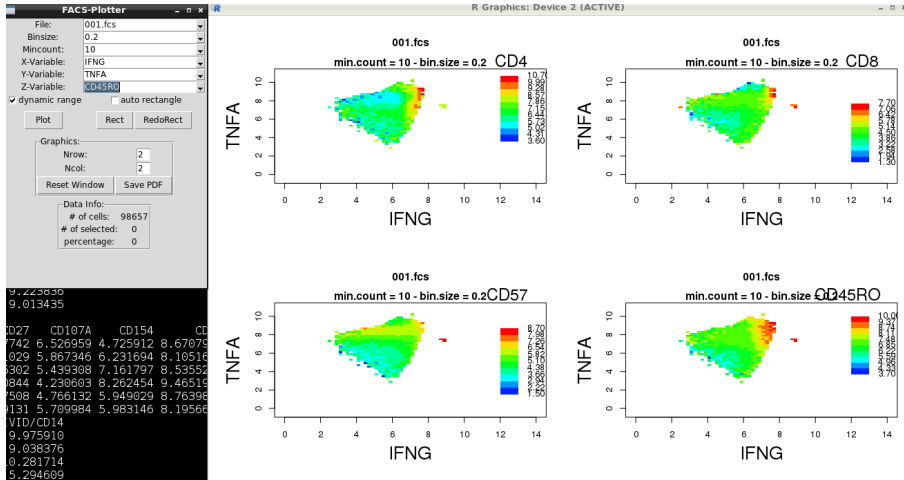
# igraph::tkplot

The following object is masked from 'package:base':

```
union
> g <- make_star(10, center=10) %U% make_ring(9, directed=TRUE)
> E(g)$width <- sample(1:10, ecount(g), replace=TRUE)
> lay <- layout_nicely(g)
>
> id <- tkplot(g, layout=lay)
> id
[1] 1
Warning message:
X11 protocol error: RenderBadPicture (invalid Picture parameter)
> tkplot.getcoords(id)
Error in eval(expr, envir, enclos) : object 'tkp.1' not found
> id <- tkplot(g, layout=lay)
> tkplot.getcoords(id)
      [,1]      [,2]
[1,] 201.99142 410.00000
[2,]  75.48916 343.69582
[3,] 116.00000 208.00000
[4,] 137.00000 118.00000
[5,] 185.73608  0.00000
[6,] 328.10755 19.14816
[7,] 424.71093 125.11378
[8,] 430.00000 267.76821
[9,] 342.08816 380.31503
[10,] 230.12706 203.30838
```



# Cytometrie-Plotter



# tk vs ttk

- tk is the old widget sets
- on Unix sometimes old looking tkscrollbar
- ttk is the newer widget set
- can be themed
- looking more modern ttkscrollbar or tk2scrollbar
- with ttk some options are missing fg, bg etc
- need to use themes, not every button should have a different color

```
> options(width=55)
> grep('ttk',ls('package:tcltk'),value=TRUE)
[1] "ttkbutton"          "ttkcheckboxbutton"  "ttkcombobox"
[4] "ttkentry"           "ttkframe"        "ttklabel"
[7] "ttklabelframe"     "ttkmenubutton"   "ttknotebook"
[10] "ttkpanedwindow"    "ttkprogressbar"  "ttkradiobutton"
```

[13]	"ttkscale"	"ttkscrollbar"	"ttkseparator"
[16]	"ttksizegrip"	"ttkspinbox"	"ttktreeview"



# Summary commands

- tktoplevel (main window)
- ttkframe, ttklabelframe, ttknotebook, ttkpanedwindow (layout)
- tkpack, tkgrid (layout manager)
- tkmenu, ttkbutton, tkentry, ttklabel and images (basic widgets)
- ttktreeview, tktext, ttkscrollbar (advanced widgets)
- tkbind and events (interaction)

# Practical Project RFenView

- a viewer for chess fen positions
- menu, file-exit, file-open (dialog)
- menu, help-about, messagebox
- frame with paned window divider
- treeview left with fen positions
- after click on entry right image label with chessboard
- package for easy access to piece images and sample fen file
- two public functions `fen2png` and `fenview` for starting the GUI

# Layout application

+-----+-----+-----+-----+			
File	(menubar)	Help	
+-----+-----+-----+-----+			
pos1			
pos2			
pos3			
...		chess position image	
	<- ->		
	(pwi)		
		(label)	
(tview)			
+-----+-----+-----+-----+			

# Programming Steps

- layout
- function outlines
- functionality
- packaging
- building, testing, releasing

# Summary

- Tcl/Tk-API usable in R as well with Python, Perl, Ruby, D, Go, ...
- How to read Tcl/Tk docs
- Widgets:
  - tkoplevel
  - t(t)kframe, ttklabelframe
  - ttknotebook, tkpanedwindow
  - ttkbutton, ttklabel, tkentry
  - tkradiobutton, ttkcheckbutton, ttkcombobox
  - tkmenu, t(t)kmenubutton
  - ttktreeview, text
  - ttkscrollbar, wrapper tkscrolled
- Layout management: pack, grid

```
> save.image('05-rtcltk.RData')  
> Stangle('../05-rtcltk.snw')
```

Writing to file 05-rtcltk.R

📁 05-rtcltk.RData 📁 05-rtcltk.R

# Links

- **Sciviews examples**

<https://web.archive.org/web/20180826195240/http://www.sciviews.org/images2/recipes-tcltk/Rtcltk.pdf>

- **old url:** <http://www.sciviews.org/recipes/tcltk/toc/>

- **Adrain Wadell:** [http://adrian.waddell.ch/EssentialSoftware/Rtcltk\\_geometry.pdf](http://adrian.waddell.ch/EssentialSoftware/Rtcltk_geometry.pdf)

- **Special issue on R-GUIs:**

<https://www.jstatsoft.org/issue/view/v049>

- **Rcmdr:** <https://socialsciences.mcmaster.ca/jfox/Misc/Rcmdr/>

- **Tk Manual Pages**

<http://www.tcl.tk/man/TkCmd/contents.htm>

## 5.10 Exercise 4 - R-tcltk

### Practical Project FastaView

- a viewer for FASTA files
- menu, file-exit, file-open (dialog)
- menu, help-about, messagebox
- frame with paned window divider
- treeview left with FASTA ids
- after click on entry text widget with sequences will be filled
- package for easy access to sample FASTA file
- public functions `read.fasta`,  
`print.FastaUtils`, `summary.FastaUtils` and  
`fastaview` for starting the GUI



# Layout application

+-----+-----+-----+		
File	(menubar)	Help
+-----+-----+-----+		
id1		
id2		
id3		
...	sequence (FASTA)	
	<- ->	
	(pwi)	
	(tktext)	
(tview)		
+-----+-----+-----+		

# Steps

## Step 1 (Layout A):

- toplevel with frame inside
- toplevel title: FenView - authorname
- tkmenu with entries File-Exit and Help-About
- simple placeholder functions:

```
OnExit = function () { print('Exit') }
```

## Step 2 (Layout B):

- ttkpanedwindow
- left add ttktreeview (1 column)
- right add ttktext
- use tkscrolled function for making them scrolled

### Step 3 (Functionality):

- file->open should return filename
- file->exit should after questioning close toplevel not exit from R
- help->about should give message about application
- use read.fasta to retrieve all Ids from FASTA file and tkinsert into treeview
- tkbind click event
- on click tkdelete old text and tkinsert wrapped sequence into text widget with FASTA header

# References

Tutorialspoint — Python 3 – Variables. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL [https://www.tutorialspoint.com/python3/python\\_variable\\_types.htm](https://www.tutorialspoint.com/python3/python_variable_types.htm). [Online; accessed 21-October-2019].

Tutorialspoint — Python 3 – Decision Making. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL [https://www.tutorialspoint.com/python3/python\\_decision\\_making.htm](https://www.tutorialspoint.com/python3/python_decision_making.htm). [Online; accessed 21-October-2019].

Tutorialspoint — Python 3 – Loops. Tutorialspoint,

SIMPLYASLEARNINMG, 2019. URL

[https://www.tutorialspoint.com/python3/python\\_loops.htm](https://www.tutorialspoint.com/python3/python_loops.htm). [Online; accessed 21-October-2019].

Tutorialspoint — Python 3 – Functions. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL

[https://www.tutorialspoint.com/python3/python\\_functions.htm](https://www.tutorialspoint.com/python3/python_functions.htm). [Online; accessed 20-October-2019].

Wikipedia — The free Encyclopedia. Wikipedia, Object-oriented programming, 2020. URL

<https://en.wikipedia.org/wiki/>

Object-oriented\_programming. [Online; accessed 27-September-2020].