# R User Group, 25.10.2004

## Lexical Scoping

- http://www.bapuetz.de/RUG/20030630/
- http://www.statistik.uni-dortmund.de/~ligges/PmitR/

Advantage of creating a separate environment: large objects do not have to be passed, i. e. no memory is wasted for creating a copy.

Ex ante annotations:
privat, parent and enclosing environment.

### Example:

```
a <- function() {
  papa <- "Wolfgang"
  both <- b()
  print(both)
}
b <- function() {
  mama <- "Herta"
  return(c(papa, mama))
}
papa    ##not found
a <- function() {
  papa <- "Wolfgang"
  a.env <- environment()    ##save current environment
  environment(b) <- a.env   ##and assign it to subroutine
  both <- b()
  print(both)
}
b <- function() {
  mama <- "Herta"
  return(c(papa, mama))
}

##print mama
a <- function() {
  papa <- "Wolfgang"
  a.env <- environment()
  environment(b) <- a.env
  both <- b()
```

```
  print(both)
  print(mama)
}
b <- function() {
  mama <- "Herta"
  return(c(papa, mama))
}
mama   ##not found

a <- function() {
  papa <- "Wolfgang"
  a.env <- environment()
  environment(b) <- a.env
  both <- b()
  print(both)
  print(mama)
}
b <- function() {
  mama <- "Herta"
  assign("mama", mama, env = a.env)
  return(c(papa, mama))
}

##nested functions: enclosing and parent environment coincide!
a <- function() {
  papa <- "Wolfgang"
  b <- function() {
    mama <- "Herta"
    return(c(papa, mama))
  }
  return(b)
}
parents <- a()
parents
parents()
papa   ##not found
ls(environment(parents))  ##list all objects defined in the
##environment of 'parents'
get("papa", envir=environment(parents))

##demonstration of global assignment
a <- function() {
```

```
  papa <<- "Wolfgang"     ##assign("papa", 2, pos = ".GlobalEnv")
  b <- function() {
    mama <- "Herta"
    return(c(papa, mama))
  }
  return(b)
}
papa   ##not found
a()
papa   ##found :-)
```

# DEBUG PACKAGE

## (I) ORDINARY DEBUGGING

**R commands:** traceback(), debug(fun), undebug(fun), browser()

**Browser commands:** ls(), where, n(ext), c(ont), recover(), Q

**dump.frames():** save current environments for later debugging

```
a <- function() {
  papa <- "Wolfgang"
  a.env <- environment()
  environment(b) <- a.env
  both <- b()
  print(both)
  print(mama)
}
a()   ##mama not found
traceback()

debug(a)
a()
##within browser
ls()                 # which objects are defined?
papa                 # what is papa
where                # current statement
n                    # execute next statement
undebug(a)

options(error = recover)
a()
# in selection: 1
# in browser:   Q

options(error = dump.frames)
a()
debugger()
# in selection: 1
# in broswer:   ls()
#               Q
```

## (II) PROFESSIONAL DEBUGGING

**Advantages:**

- setting breakpoints
- listing functions that are in debug mode

**Disadvantage:**

- dependency on tcl/tk and mvbutils packages, i. e. R has to be installed by shell commando: .configure --with-tcl-config=/usr/lib/tclConfig.sh --with-tk-config=/usr/lib/tkConfig.sh

**Customize debugging window in .First() of .Rprofile:**

```
library(debug)
##do not print objects > 10 bytes
options(threshold.debug.autoprint.size=100)
options(debug.height=30)   ##debug window height (in lines)
options(debug.width=60)    ##debug window width (in chars)
options(debug.font="Times 14")
options(debug.command.recall=F)
```

**Documentation:** R–1.9.1/library/debug/html/00README.debug.html
Calling a mtraced function evokes a debugging window containing a copy of the code with line numbers and the debugging mode with prompt 'D(...)>'. Current line to execute is highlighted.

**Commands:**

| | | |
|---|---|---|
| mtrace(fun) | $\Longrightarrow$ | turn on debugging mode for function 'fun' |
| mtrace(fun, tracing=F) | $\Longrightarrow$ | turn off debugging mode for function 'fun' |
| mtrace.off() | $\Longrightarrow$ | turn off debugging mode for all functions |
| names(tracees) | $\Longrightarrow$ | list all currently mtraced functions |
| <ENTER> | $\Longrightarrow$ | stepwise execution |
| go(), go(line.no) | $\Longrightarrow$ | block-wise execution |
| skip(line.no) | $\Longrightarrow$ | skip execution |
| bp(line.no) | $\Longrightarrow$ | breakpoints is marked with an asteriks and there is always one in the first line |
| bp(line.no, F) | $\Longrightarrow$ | clearing breakpoint |
| bp(line.no, index > 3) | $\Longrightarrow$ | conditional breakpoint |
| bp(line.no, {var <- 3; FALSE}) | $\Longrightarrow$ | subsequent manipulation of a variable value (non-breaking breakpoint) |
| tracees$f$breakpoints | $\Longrightarrow$ | list active breakpoints of function f |
| qqq() | $\Longrightarrow$ | quit debugger |

CAVEAT: Pasting any code containing linebreaks into the debugging mode will R cause to crash!

EXAMPLE:

```
a <- function() {
  papa <- "Wolfgang"
  a.env <- environment()
  environment(b) <- a.env
  both <- b()
  print(both)
  print(mama)
}
b <- function() {
  mama <- "Herta"
  assign("mama", mama, env = a.env)
  return(c(papa, mama))
}
library(debug)
mtrace(a)
names(tracees)
tracees         ##expr, breakpoint, line.list, n
mtrace(b)
names(tracees)

a()             ##debugging mode
go(4)
<ENTER>         ##step into subroutine
go(4)
skip(7)         ##skip printing
mtrace.off()

##extension to a for-loop
a <- function() {
  papa <- "Wolfgang"
  a.env <- environment()
  environment(b) <- a.env
  for (i in 1:10) {
    both <- b()
    print(both)
  }
}

mtrace(a)
tracees$a$line.list
bp(5, i == 7, fname="a")
```

```
tracees$a$breakpoint
a()                     ##stops at first statement
go()                    ##continue till conditional breakpoint holds
papa <- "Bernd"         ##manipulate variable
go()
bp(5, F, "a")

##non-breaking breakpoint: variable patching
bp(5, {cat(paste("\n", i, "years of lucky marriage:")); FALSE},
   fname="a")
tracees$a$breakpoint
a()
```

## RWinEdt

**Reference:** http://cran.r-project.org/contrib/extra/winedt/

**Features:** syntax–highlighting, parenthesis matching, buttons/shortcuts for simple save/source/submit of R–files and selected code.

## WinEdt

- Line/Block text selection
- pdflatex/acrobat reader icons
- mathsymbols via $\Sigma$–icon
- 'set main file'–icon for compilation of main.tex when include1.tex, include2.tex etc are open at the same time.