An Introduction to Programming in R

Drew Schmidt

November 13, 2015

Notes

About

This tutorial:

- This is not a "how to fit a linear model in R" tutorial.
- Slides and exercises are available at: http://wrathematics.info/handouts/bas2015.html
- I try to stick to base R as much as possible, even when better alternatives exist.
- Mostly independent; we can skip something that's boring!

Me:

- I am a very productive R package developer.
- I mostly do not show you the things I make.
- My expertise is R in HPC (contact me if interested!)

Notes

Notes

What is R?

Notes

What is R?

lingua franca for data analytics and statistical computing.

Part programming language, part data analysis package.

Dialect of S (Bell Labs).

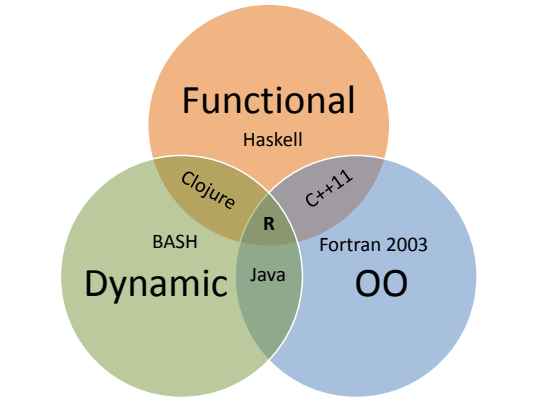
Syntax designed for data.

Notes

Who uses R?

Notes

Language Paradigms



Notes

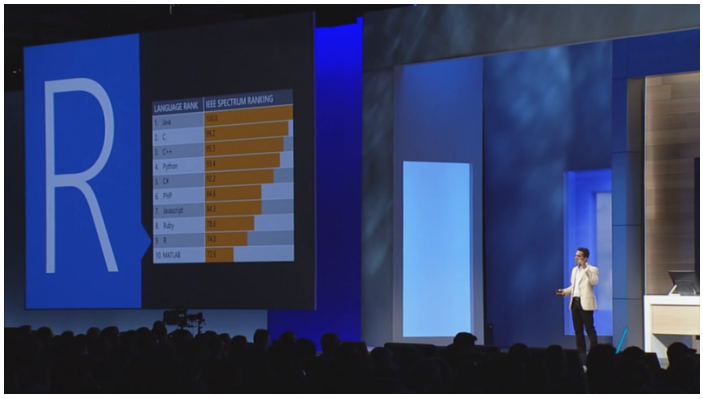
Very Popular for a DSL!

IEEE Spectrum's 2014 Ranking of Programming Languages

Language Rank	Types	Spectrum Ranking
1. Java		100.0
2. C		99.3
3. C++		95.5
4. Python		93.4
5. C#		92.4
6. PHP		84.7
7. Javascript		84.4
8. Ruby		78.8
9. R		74.2
10. MATLAB		72.9

See:
http://spectrum.ieee.org/static/interactive-the-top-programming-languages#index

Notes



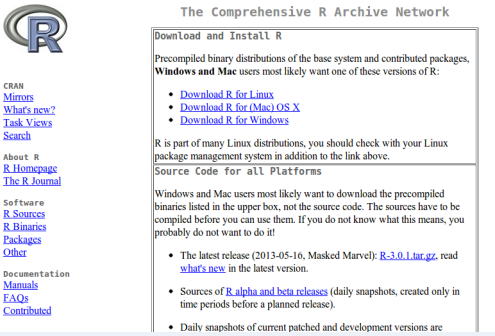
At Build 2015 Microsoft CVP Joseph Sirosh called R the "language of data" and said *"if there is a single language that you choose to learn today .. let it be R"*.

Notes

Notes

Installing R

Go to http://cran.r-project.org/ for a download (binary or source)



Notes

Installing R

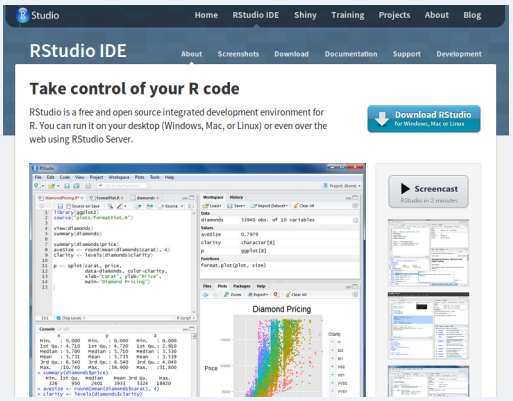
Windows users should also install Rtools
http://cran.r-project.org/bin/windows/Rtools/

For a complete set of installation instructions, see
http://www.r-pbd.org/install.html

Notes

Installing Rstudio

Go to http://www.rstudio.com/ide/



Notes

Notes

Important things we can't cover

- R and version control
- Developing R packages
- Performance/profiling
- Graphics/visualization

Notes

R Resources

The Art of R Programming by Norm Matloff:
http://nostarch.com/artofr.htm

An Introduction to R by Venables, Smith, and the R Core Team:
http://cran.r-project.org/doc/manuals/R-intro.pdf

The R Inferno by Patrick Burns:
http://www.burns-stat.com/pages/Tutor/R_inferno.pdf

Mathesaurus: http://mathesaurus.sourceforge.net/

R programming for those coming from other languages: http://www.johndcook.com/R_language_for_programmers.html

aRrgh: a newcomer's (angry) guide to R, by Tim Smith and Kevin Ushey: http://tim-smith.us/arrgh/

Notes

Tutorials

R Programming, Coursera course through Johns Hopkins
https://www.coursera.org/course/rprog

Statistics One Coursera course through Princeton
https://www.coursera.org/course/stats1

High Performance Computing with R
https://github.com/wrathematics/2015hpcRworkshop/blob/master/README.md

Notes

Other Invaluable Resources

R Installation and Administration:
http://cran.r-project.org/doc/manuals/R-admin.html

Task Views: http://cran.at.r-project.org/web/views

Writing R Extensions:
http://cran.r-project.org/doc/manuals/R-exts.html

Mailing list archives: http://tolstoy.newcastle.edu.au/R/

The [R] stackoverflow tag.

The #rstats hastag on Twitter.

Notes

Comments and Advice

R is part statistics package, part programming language.

R is slow; if you don't know what you're doing, it's *really* slow.

There is an R help mailing list. Use stackoverflow instead....

Learn to love the R help system.

If something appears broken in core R, it's (probably) not them, it's you.

Try to avoid "super functions".

Notes

Notes

Notes

Interacting with the R Terminal

The default thing for R to do is print/show:

```
1
1+1
print(1+1)
sum

+ # ctrl+c to break
"+"
`+`
```


Notes

Arithmetic

```
7+4
7-4
7*4
7/4
7^4
7%%4
```


Notes

Assignment

R naming rules can be quite lax. For all practical purposes:

- Start with a letter
- Should consist of letters (any case), numbers, .'s, and _'s
- Very strange things are possible, however...

```
rm(list=ls())
```

```
"&*()" <- 3

"2" <- 1
`2` + 1
## [1] 2

"\U0001f431" <- "even unicode"
cat(ls())
## &*() 🐱 2
```


Notes

Assignment

```
# Local
myvar <- 1
myvar = 4

# Global
myvar <<- 2

# Whever you want
assign(x="myvar", value=3)
```


Notes

Case and Spacing

R is case sensitive, but fairly lax about spacing:

```
x <- 1:5
x
X

1      +  1
sum(x)
sum ( x     )
s um(x)
```


Notes

Finding Help

R has its own manual system.

Most of the answers to your questions lie within.

Find help using ? or help(), or search across all help with ??.

```
?sum
??sum
help("sum")

?+
?"+"
```


Notes

Ways to Run R

Interactive: typing commands into the console.

Batch: Rscript, R CMD BATCH.

From an IDE/GUI.

Batch from a GUI: source()

Notes

On the note of directories...

Windows paths use / or
, e.g. c:/myfile.r

Get current working directory: getwd()

Set current working directory: setwd("path/to/my/dir")

Notes

Notes

R Extensions

R is great, but limited.

Has a great package extension system.

R doesn't do what you want? Make it!

"*...if I don't know how to fix it, I can hire somebody else to fix it for me.*"
— Matt Dowle, developer of the data.table package.

Notes

Packages

"R extensions"

Comprehensive R Archive Network (CRAN).

7474 packages (nrow(available.packages()))

CRAN Task Views http://cran.r-project.org/web/views/

Notes

Terminology

A package is a collection of code usable by R.

A library is a collection of packages.

library() loads a package.

Don't think too much about this...

Notes

Terminology

We call the extension a *package*.

Packages go into a *library*.

This is dumb and confusing, but there's nothing we can ever do about it.

Notes

Example Confusion

I wrote a library.

I put the library in a package.

I install the package ...into a library.

I load the package with library() ???

Notes

BOOM

Notes

Installing R Packages

```
install.packages("devtools")
```

```
install.packages("devtools", lib="some/place/on/disk")
```

```
R CMD INSTALL devtools_1.6.tar.gz -l /some/place/on/disk
```


Notes

Repositories

- This basically assumes you're using CRAN.
- Lots of exciting development is happening outside of CRAN these days.
- Other binary package repositories: Bioconductor, R-forge (Windows)
- Other source repositories: GitHub, Bitbucket, ...

Notes

GitHub Binaries?

Notes

Installing Packages from Source

To install packages from source, you need some compilers:

Windows: Install Rtools.

Mac: Install Xcode from the app store, possibly some other things from here. If you need OpenMP, god help you.

Linux and FreeBSD: You're good to go.

Notes

Installing R Packages

```
devtools::install_github("wrathematics/lineSampler")
```

Devtools Package Install Functions

install_bitbucket	install_github	install_svn
install_deps	install_gitorious	install_url
install_git	install_local	install_version

Notes

Notes

I/O

Input	Output
readcsv()	writecsv()
readtable()	writetable()
readBin()	writeBin()
readLines()	writeLines()
save()	load()
saveRDS()	readRDS()
scan()	

Notes

"What about my really weird use-case?"

NUMEROUS R packages

XML, json, databases, ...

See also: *R Data Import/Export* manual https://cran.r-project.org/doc/manuals/r-release/R-data.html

Notes

Reading and Writing a CSV

```
x <- matrix(1:30, 10)
write.csv(x, file="x.csv")
read.csv("x.csv")

write.csv(x, file="x.csv", row.names=FALSE)
read.csv("x.csv", header=TRUE)
```


Notes

The dreaded stringsAsFactors

Default for R functions (e.g., read.csv()) is stringsAsFactors=TRUE.

If you do not need to modify the "strings" (going straight to modeling), stringsAsFactors=TRUE.

If you **DO** need to modify, stringsAsFactors=FALSE

Notes

Serializing

```
x <- letters
y <- 1:5
z <- list(list("a"), b=matrix(0))

save(x, y, z, file="robjects.RData")
rm(x)
rm(y)
rm(z)
x
load("robjects.RData")
```

See also saveRDS().

Notes

Why/Why Not Serialize?

Why:

- Serializing is a binary "as-is" format.
- Performance!

Why not:

- Only works with R (or something that can read R binary formats!)
- Endianness...

Notes

Some Notable Packages

General:

- rio https://cran.r-project.org/web/packages/io/index.html

CSV and friends:

- data.table (fread()) https://cran.r-project.org/web/packages/data.table/index.html
- readr https://cran.r-project.org/web/packages/readr/index.html

Readers of proprietary things:

- haven (SAS, SPSS, and STATA) https://cran.r-project.org/web/packages/haven/index.html
- readxl (Excel) https://cran.r-project.org/web/packages/readxl/index.html

Notes

The data() Command

- Many packages (and R itself) bundle data.
- Load data with data()
- For a list of R's: library(help="datasets")

Notes

Example

```
head(iris)
head(mtcars)

data(package="ggplot2")
diamonds # error
data(diamonds)
head(diamonds)
```


Notes

Notes

Strings

"character" data (text).

Internal storage scheme is complicated...

Managing character data is its own course!

Just the baby basics...

Notes

Quotes

Three kinds: single quote ', double quote ", backtick `.

Single and double create strings.

Backticks access language objects (e.g., `+`).

Escape within a string with a *single backslash*:

```
quoted_quote <- "\"\n"
quoted_quote
## [1] "\"\n"
cat(quoted_quote)
## "
```


Notes

Example 1

```
letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
    "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"

toupper(letters)
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
    "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"

LETTERS
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
    "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"

tolower(LETTERS)
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
    "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```


Notes

Example 2

```
x <- "Star Trek is objectively better than Star Wars"
strsplit(x, split=" ")
## [[1]]
## [1] "Star"        "Trek"        "is"          "objectively"
    "better"
## [6] "than"        "Star"        "Wars"

y <- unlist(strsplit(x, split=""))
y
##  [1] "S" "t" "a" "r" " " " " "T" "r" "e" "k" " " " " "i" "s" " " " " "o" "b"
    "j" "e" "c" "t"
## [20] "i" "v" "e" "l" "y" " " " " "b" "e" "t" "t" "e" "r" " " " " "t" "h"
    "a" "n" " " " " "S"
## [39] "t" "a" "r" " " " " "W" "a" "r" "s"

paste(rev(y), collapse="")
## [1] "sraW ratS naht retteb ylevitcejbo si kerT ratS"
```


Notes

Example 3

```
paste(letters, letters)
##  [1] "a a" "b b" "c c" "d d" "e e" "f f" "g g" "h h" "i i" "j j"
    "k k" "l l"
## [13] "m m" "n n" "o o" "p p" "q q" "r r" "s s" "t t" "u u" "v v"
    "w w" "x x"
## [25] "y y" "z z"

paste(letters, letters, sep="")
##  [1] "aa" "bb" "cc" "dd" "ee" "ff" "gg" "hh" "ii" "jj" "kk" "ll"
    "mm" "nn" "oo"
## [16] "pp" "qq" "rr" "ss" "tt" "uu" "vv" "ww" "xx" "yy" "zz"

paste(letters, letters, sep="", collapse="")
## [1] "aabbccddeeffgghhiijjkkllmmnnooppqqrrsttuuvvwwxxyyzz"

paste(paste(letters, collapse=""), paste(LETTERS, collapse=""),
    sep="")
## [1] "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```


Notes

Example 4

```
x <- rnorm(1000, mean=10)

paste("The mean of 'x' is:", mean(x))
## [1] "The mean of 'x' is: 10.0157377724829"

cat(sprintf("mean:\t%.2f\nvar:\t%.2f\n", mean(x), var(x)))
## mean:    10.02
## var:     0.95
```


Notes

Regular Expressions

grep(), grepl()

sub(), gsub()

regexpr(), gregexpr()

Some others...

Notes

Some Notable Packages

stringi https://cran.r-project.org/web/packages/stringi/index.html

tm https://cran.r-project.org/web/packages/tm/index.html

Notes

Notes

Dates

Dates managed in the UNIX style.

Probably very alien for you...

Just use lubridate.

Notes

Formatting Dates with Lubridate

```
rightnow <- Sys.time()
rightnow

library(lubridate)
year(rightnow)
## [1] 2015

month(rightnow)
## [1] 11
month(rightnow, label=TRUE)
## [1] Nov

wday(rightnow, label=TRUE)
## [1] Mon
```


Notes

And much more!

See the package vignette: https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html

Notes

Example: Timezone Lookup

```
timezone <- function()
{
  time <- Sys.time()
  ret <- list(timezone=format(time, format="%Z"),
      UTC.offset=format(time, format="%z"))
  class(ret) <- "tzlookup"
  return(ret)
}

print.tzlookup <- function(x)
{
  maxlen <- max(sapply(names(x), nchar))
  spacenames <- c("timezone:     ", "UTC Offset:")
  cat(paste(spacenames, x, sep=" ", collapse="\n"), "\n")
}

timezone()
## timezone:      EST
## UTC Offset: -0500
```


Notes

Some Notable Packages

lubridate https://cran.r-project.org/web/packages/lubridate/index.html

Notes

Notes

Motivation

For many, dataframes are the most important/common object.

ENORMOUS topic.

We cover some important techniques.

Notes

Modifying Elements

```
### Matrix-like notation
iris[1,1] <- 0
iris[, 1] <- 0

edit(iris)
```


Notes

Adding Variables

```
index <- 1:nrow(iris)
iris$index <- index
head(iris)

rm(iris)
iris <- cbind(iris, index)
head(iris)

rm(iris)
iris[, ncol(iris)+1] <- index
head(iris)

rm(iris)
```


Notes

Adding Rows

```
nrow(iris)
set.seed(12345)
newrow <- iris[sample(nrow(iris), size=1), ]

iris <- rbind(iris, newrow)
nrow(iris)

iris[nrow(iris)+1, ] <- newrow
nrow(iris)
```


Notes

Adding Variables

```
index <- 1:nrow(iris)
iris$index <- index
head(iris)

rm(iris)
head(iris)

iris <- cbind(iris, index)
head(iris)

rm(iris)
```


Notes

Subsetting

```
subset(iris, Sepal.Length > 7)

subset(iris, Sepal.Length > 7 & Petal.Length < 6)

subset(iris, Species == "setosa" | Species == "versicolor")
```


Notes

Sorting/Ordering Rows

```
df1 <- iris[order(iris$Sepal.Length), ]
head(df1)

df2 <- iris[order(-iris$Sepal.Length), ]
head(df2)

df3 <- iris[order(-iris$Sepal.Length, iris$Sepal.Width), ]
head(df3)
```


Notes

Sorting/Ordering Columns

```
iris <- iris[1:5, ]

iris[5:1]
set.seed(12345)
iris[sample(1:ncol(iris))]

iris[1]

iris[sort(colnames(iris))]

iris[sort(colnames(iris), decreasing=TRUE)]

rm(iris)
```


Notes

Dealing with Duplicates

```
any(duplicated(iris))
which(duplicated(iris))

nrow(unique(iris))
```


Notes

Applying Functions

```
colMeans(iris)
colMeans(iris[, -5])

apply(iris[, -5], MARGIN=2, FUN=median)
```


Notes

Some Notable Packages

Dataframe-like things:

- data.table https://cran.r-project.org/web/packages/data.table/index.html
- dplyr https://cran.r-project.org/web/packages/dplyr/index.html

Restructuring helpers:

- reshape2 https://cran.r-project.org/web/packages/reshape2/index.html
- tidyr https://cran.r-project.org/web/packages/tidyr/index.html
- broom https://cran.r-project.org/web/packages/broom/index.html

Notes

Notes

Notes

The R Language

Storage: logical, int, double, double complex, character (strings)

Structures: vector, matrix, array, list, dataframe, environment (hashtable)

Caveats: (Logical) TRUE, FALSE, NA

In fact, there's an NA for each type:

Integer: $-(2^{-31}-1)$
Double: value at address 0x7FF00000000007A2LL

3 official OOP systems, several unofficial ones.

Several unofficial packages supporting C++.

Notes

Data Types

- logical
- int
- double
- double complex
- character

Notes

Data Types

R is *dynamically typed*. You do not have to declare what kind of data a variable is before you start using it:

```
x <- 1
typeof(x)
x <- 1:2
typeof(x)
```


Notes

Other

There are 4 "other" data "types":

- Inf
- NaN
- NULL
- NA

Notes

Inf

Numerical infinity

```
Inf
## [1] Inf

typeof(Inf)
## [1] "double"

is.finite(Inf)
## [1] FALSE

is.infinite(Inf)
## [1] TRUE

Inf+Inf
## [1] Inf

1/0
## [1] Inf
```


Notes

NaN

Not a Number; numerical undefinedness.

```
NaN
## [1] NaN

typeof(NaN)
## [1] "double"

is.nan(NaN)
## [1] TRUE

Inf-Inf
## [1] NaN

sin(Inf)
## [1] NaN
## Warning message:
## In sin(Inf) : NaNs produced
```


Notes

NULL

The null object; a sort of placeholder for something undefined. Like a non-numeric NaN.

```
NULL
## NULL

typeof(NULL)
## [1] "NULL"

is.null(NULL)
## [1] TRUE

NULL+NULL
## numeric(0)
```


Notes

NA

Missingness; not merely undefined, but *unknown*.

- Each type (logical, int, double, ...) has its own NA
- R is thus not boolean: TRUE, FALSE, NA
- Most R methods have ways of removing NA's

Notes

Data Structures

- vector
- matrix
- array
- factor
- list
- dataframe

Notes

Vectors

```
1:10
10:1

c(1, 3, 5, 7, 9)
seq(from=1, to=10, by=2)

x <- 1:5
length(x)
```


Notes

Matrices

```
matrix(1:10)
matrix(1:10, nrow=5)
matrix(1:10, ncol=5)

x <- 1:10
y <- as.matrix(x)
dim(x)
dim(y)
dim(x) <- c(1, 10)
x
```


Notes

Extraction

```
x <- matrix(1:30, 10)
x

x[-1, ]
x[, -1]
x[1:5, -1]
x[c(2,5,7), c(1,3)]

y <- x[, -2]
dim(y) <- NULL
y
```


Notes

Replacement

```
x <- matrix(1:30, 10)

x[1:5, ] <- 0
x[7, 3] <- NA
x
```


Notes

Factors

```
factor(1:5)
factor(c("a", "b", "b", "a", "c"))

x <- factor(-1:1)
x
as.numeric(x)
as.numeric(as.character(factor(-1:1)))
```


Notes

Dataframes

```
c(1, "a")
matrix(c(1, "a"))

x <- data.frame(1, "a")
x
x[1, 1]
is.numeric(x[1,1])
x[1, 2]
is.numeric(x[1,2])

data.frame(a=1:5,b=5:1)
```


Notes

Lists

- Super structures
- Items can be any structure (even other lists)
- Dataframe is really just a special list

Notes

Lists

```
list(1)
x <- list(list(1), "a")
x
x[[1]]

x <- list(a=list("b", 1), z=1:5)
x$z
```


Notes

Other Structures?

stacks, heaps, queues, graphs, ...

tl;dr: mostly no

Hash table - environments

Example deque https://github.com/wrathematics/dequer

Notes

Notes

Logic

Possible values are TRUE, FALSE, and NA

Operations: &&, ||, &, |

Comparators: ==, !=, <, <=, >, and >=

Notes

Logic

```
1==1
1!=1
1<1
1<=1

TRUE==FALSE

TRUE==1
FALSE==0

TRUE==T
FALSE==F
```


Notes

Logic

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

NULL==NULL
is.null(NULL)

NaN==NaN
is.nan(NaN)

Inf==Inf
is.infinite(Inf)
```


Notes

Logic

Vectors of logicals are evaluated element-wise:

```
x <- c(TRUE, FALSE, FALSE, TRUE)
x==TRUE
!x
```


Notes

Logic

Some very important functions for logical evaluations:

```
x <- c(TRUE, FALSE, FALSE, TRUE)
any(x)
all(x)
which(x)

y <- -5:5
which(y%%2==0)
y[which(y%%2==0)]
```


Notes

Comparisons

Comparing to...

- Comparing to NULL: is.null(x)
- Comparing to NA: is.na(x)
- Comparing to TRUE: isTRUE(x)

Comparing two...

- Tread carefully...: x == y
- Numerics: all.equal(x, y)
- EXACTLY THE SAME: identical(x, y)

Notes

Why all this trouble?

BECAUSE COMPUTERS ARE TERRIBLE

```
x <- 1
for (i in 1:10) x <- x - .1
x
```


Notes

Another famous example

```
sprintf("%.17f", 0.1+0.2)
## [1] "0.30000000000000004"
```

This even has its own website! http://0.30000000000000004.com/

Notes

Notes

Loops

Instruction set to be repeatedly executed until some condition is satisfied (possibly forever).

R has for() and while().

for(): Iterates over a list or vector

while(): Performs operations until logical condition is satisfied.

Notes

for Loop

```
for (i in 1:10){
  cat(i, " ")
}
## 1  2  3  4  5  6  7  8  9  10

x <- matrix(1:30, 10)
colmax <- numeric(3)
for (i in 1:3){
  colmax[i] <- max(x[, i])
}

colmax
## [1] 10 20 30
```


Notes

while Loop

```
i <- 1
while (i < 11){
  print(i)
  i <- i+1
}

n <- 2
i <- 1
while (n < 1000){
  n <- n^2
  i <- i*2
}

n
## [1] 65536
i
## [1] 16
```


Notes

The *ply Family

apply(): Apply function across "margin" (dimension) of matrix.
lapply(): Apply function to input data object; returns a list.
sapply(): Same as sapply() but
mapply(): Multivariate sapply().
vapply(): Essentially sapply(), sometimes faster.
tapply(): Applying a function to a subset of a vector.
...

We will only discuss the first 3.

Notes

What is it?

Syntax to loop without writing a loop.
Inspired by functional programming.
Not the same thing as vectorization (but it looks vectorized).

Notes

apply

```
x <- matrix(1:30, 10)
x

apply(X=x, MARGIN=1, FUN=min)
## [1]  1  2  3  4  5  6  7  8  9 10
apply(X=x, MARGIN=2, FUN=min)
## [1]  1 11 21

out <- numeric(3)
for (i in 1:3){
  out[i] <- min(x[, i])
}
out
## [1]  1 11 21
```


Notes

lapply and sapply

```
x <- 1:5
lapply(X=x, FUN=sqrt)
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1.414214
##
## [[3]]
## [1] 1.732051
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 2.236068

sapply(X=x, FUN=sqrt)
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```


Notes

*ply Functions Internally

apply(): A for() loop.

lapply(): Internal R voodoo; faster than a loop.

sapply(): Essentially the same as lapply().

Notes

When does all this choice matter?

loops are slow.

apply() is sugar for an R for loop.

lapply() different, often faster.

Vectorization is fastest of all.

Notes

Loop Speeds

```
x <- 100000:1

system.time({ # No initialization
  sin <- numeric(0)
  for (i in 1:length(x)){
    sin[i] <- sin(x[i])
  }
  sin
})
##    user  system elapsed
## 17.320   1.072  18.376

system.time({ # With initialization
  sin <- numeric(length(x))
  for (i in 1:length(x)){
    sin[i] <- sin(x[i])
  }
  sin
})
##    user  system elapsed
##   0.172   0.000   0.171
```


Notes

Loop Speeds

```
system.time(lapply(x, sin))
##  user  system elapsed
## 0.056   0.004   0.059

system.time(sapply(x, sin))
##  user  system elapsed
## 0.048   0.004   0.053

system.time(sin(x))
##  user  system elapsed
## 0.004   0.000   0.004
```


Notes

Notes

Functions

Self-contained input/output machine.

Reusable blocks of code.

First class objects.

All functions have returns!

Evaluating the Design of the R language, http://r.cs.purdue.edu/pub/ecoop12.pdf

Notes

Some Basic Rules

Can not modify data in place* (multi-return with a list)

Arguments can have defaults, specified with =.

By default, no printing occurs (have to say print(x)).

The last "thing done" is the return.

People often use NULL or invisible(NULL) if return is unimportant.

Notes

Functions: Example 1

```
f <- function(x)
{
  ret <- x+1
  return(ret)
}

f(1)
## [1] 2
f(2)
## [1] 3
f(5)
## [1] 6
f
## function(x)
## {
##   ret <- x+1
##   return(ret)
## }
```


Notes

Functions: Example 2

```
f <- function (a, b)
{
  a - b
}

f(a=1, b=2)
f(1, 2)
f(b=1, a=2)
f(b=1, 2)
f(1)
f(matrix(1:4, nrow=2), matrix(4:1, nrow=2))
```


Notes

Functions: Example 3

For complicated returns (especially of mixed type/class), use a list:

```
g <- function (a, b)
{
  plus <- a+b
  minus <- a-b
  return(list(plus, minus))
}

g(5, 2)
g(1, 0)
g(f(2, 6), 2)
```


Notes

Functions: Example 4

R allows parameter defaults in functions

```
h <- function (a=1, b=2)
{
  return(b-a)
}

h
h()
h(2, 1)
```


Notes

Recursive Functions

A *recursive function* is one that calls itself:

```
f <- function(n)
{
  if (n==1)
    return(1)
  else {
    if (n%%2==0)
      return(f(n/2))
    else
      return(f(3*n+1))
  }
}

f(22)
## [1] 1
f(237)
## [1] 1
sapply(1:37, f)
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    1 1 1 1 1 1 1
```


Notes

Notes

Debugging R Code

Very broad topic ...

We'll hit the highlights.

For more examples, see:
cran.r-project.org/doc/manuals/R-exts.html#Debugging

Debugging compiled code called by R (valgrind, gdb, ...) also possible...

Notes

Object Inspection Tools

print()
str()
unclass()

Notes

Object Inspection Tools: print()

Basic printing:

```
x <- matrix(1:10, nrow=2)
print(x)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10

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


Notes

Object Inspection Tools: str()

Examining the **str**ucture of an R object:

```
x <- matrix(1:10, nrow=2)

str(x)
##  int [1:2, 1:5] 1 2 3 4 5 6 7 8 9 10
```


Notes

Object Inspection Tools: unclass()

Exposing all data with unclass():

```
df <- data.frame(x=rnorm(10), y=rnorm(10))
mdl <- lm(y~x, data=df) ### That's a "tilde" character

mdl
print(mdl)

str(mdl)

unclass(mdl)
```


Notes

The R Debugger

debug()
debugonce()
undebug()

Notes

Using The R Debugger

1. Declare function to be debugged: debug(foo)
2. Call function: foo(arg1, arg2, ...)
   - next: Enter or n followed by Enter.
   - break: Halt execution and exit debugging: Q.
   - exit: Continue execution and exit debugging: c.
3. Call undebug() to stop debugging

Notes

Using the Debugger

Example Debugger Interaction

```
> f <- function(x){y <- z+1;z <- y*2;z}
> f(1)
Error in f(1) : object 'z' not found
> debug(f)
> f(1)
debugging in: f(1)
debug at #1: {
    y <- z + 1
    z <- y * 2
    z
}
Browse[2]>
debug at #1: y <- z + 1
Browse[2]>
Error in f(1) : object 'z' not found
>
```


Notes

Notes

Thanks so much for attending!

Questions?

Notes