Introduction to R

v2019-01

Babraham
Bioinformatics

R can just be a calculator

```
> 3+2
[1] 5

> 2/7
[1] 0.2857143

> 5^10
[1] 9765625
```

Babraham Bioinformatics

Storing numerical data in variables

```
10 -> x
y <- 20

x
[1] 10

x/y
[1] 0.5

x/y -> z
```

Babraham
Bioinformatics

Storing text in variables

```
my.name <- "laura"

my.other.name <- 'biggins'
```

Babraham
Bioinformatics

Running a simple function

```
sqrt(10)
[1] 3.162278
```

Babraham
Bioinformatics

Looking up help

```
?sqrt
```

MathFun {base} R Documentation

Miscellaneous Mathematical Functions

Description

`abs(x)` computes the absolute value of x, `sqrt(x)` computes the (principal) square root of x, $\sqrt{x}$.

The naming follows the standard for computer languages such as C or Fortran.

Usage

```
abs(x)
sqrt(x)
```

Arguments

`x` a numeric or `complex` vector or array.

Details

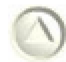
These are internal generic primitive functions: methods can be defined for them individually or via the `Math` group generic. For complex arguments (and the default method), `z`, `abs(z) == Mod(z)` and `sqrt(z) == z^0.5`.

`abs(x)` returns an `integer` vector when `x` is `integer` or `logical`.

Searching Help

Search Results

??substring



Help pages:

Biostrings::class:MultipleAlignment	MultipleAlignment objects
Biostrings::class:XString	BString objects
Biostrings::class:XStringSet	XStringSet objects
Biostrings::class:XStringSetList	XStringSetList objects
Biostrings::class:XStringViews	The XStringViews class
Biostrings::letterFrequency	Calculate the frequency of letters in a biological sequence, or the consensus matrix of a set of sequences
Biostrings::longestConsecutive	Obtain the length of the longest substring containing only 'letter'
Biostrings::lcprefix	Longest Common Prefix/Suffix/Substring searching functions
Biostrings::extractAt	Extract/replace arbitrary substrings from/in a string or set of strings.
crayon::col_substr	Substring(s) of an ANSI colored string
crayon::col_substring	Substring(s) of an ANSI colored string
Hmisc::makeNstr	creates a string that is a repeat of a substring
Hmisc::sedit	Character String Editing and Miscellaneous Character Handling Functions
S4Vectors::Rle-utils	Common operations on Rle objects
stringi::stri_sub	Extract a Substring From or Replace a Substring In a Character Vector
stringr::str_sub	Extract and replace substrings from a character vector.
base::regmatches	Extract or Replace Matched Substrings
base::substr	Substrings of a Character Vector

Searching Help

substr {base}

R Documentation

Substrings of a Character Vector

Description

Extract or replace substrings in a character vector.

Usage

```
substr(x, start, stop)
substring(text, first, last = 1000000L)
substr(x, start, stop) <- value
substring(text, first, last = 1000000L) <- value
```

Arguments

`x, text`	a character vector.
`start, first`	integer. The first element to be replaced.
`stop, last`	integer. The last element to be replaced.
`value`	a character vector, recycled if necessary.

Passing arguments to functions

```
substr(my.name,2,4)
[1] "aur"

substr(x=my.name,start=2,stop=4)
[1] "aur"

substr(
  start=2,
  stop=4,
  x=my.name
)
[1] "aur"
```

Babraham
Bioinformatics

Exercise 1

Everything is a vector

- Vectors are the most basic unit of storage in R

- Vectors are ordered sets of values of the same type
  - Numeric
  - Character (text)
  - Factor
  - Logical
  - Date etc…

```
10 -> x
```

x is a vector of length 1 with 10 as its first value

Babraham Bioinformatics

Creating vectors manually

- Use the "c" (combine) function

```
c(1,2,4,6,3) -> simple.vector

c("simon","laura","anne","jo","steven") -> some.names
```

- Data should be of the same type

```
c(1,2,3,"fred")
[1] "1"    "2"    "3"    "fred"
```

Babraham Bioinformatics

Functions for creating vectors

- `rep` - repeat values

```
rep(2,10)
 [1] 2 2 2 2 2 2 2 2 2 2

rep("hello",5)
[1] "hello" "hello" "hello" "hello" "hello"

rep(c("dog","cat"),times=3)
[1] "dog" "cat" "dog" "cat" "dog" "cat"

rep(c("dog","cat"),each=3)
[1] "dog" "dog" "dog" "cat" "cat" "cat"
```

Functions for creating vectors

- `seq` - create numerical sequences
  - No required arguments!
    - `from`
    - `to`
    - `by`
    - `length.out`
  - Specify enough that the series is unique

Babraham Bioinformatics

Functions for creating vectors

- `seq` - create numerical sequences

```
seq(from=2,by=3,to=14)
[1]  2  5  8 11 14

seq(from=3,by=10,to=40)
[1]  3 13 23 33

seq(from=5,by=3.6,length.out=5)
[1]  5.0  8.6 12.2 15.8 19.4
```

Functions for creating vectors

- Sampling from statistical distributions
  - `rnorm`
  - `runif`
  - `rpois`
  - `rbeta`
  - `rbinom`

```
rnorm(10000)
```

Babraham Bioinformatics

Language shortcuts for vector creation

- Single elements

```
"simon"
c("simon")
```

- Integer series

```
seq(from=4,to=20,by=1)
4:20
```

Viewing large variables

- In the console

```
head(data)
tail(data,n=10)
```

- Graphically

```
View(data)
```

[Note capital V!]

Click in Environment tab

Babraham Bioinformatics

What can we do with Vectors?

- Extract subsets
- Perform vectorised operations

- Both are *really* useful!

Babraham
Bioinformatics

Extracting from a vector

- Always two ways to retrieve data from an R data structure

1. Based on its position (give me the third value)
2. Based on a name (give me the BRCA1 value)

- True for all of the main R structures

Babraham
Bioinformatics

Extracting by position

```
simple.vector
[1] 1 2 4 6 3

simple.vector[5]
[1] 3

simple.vector[c(5,2,3)]
[1] 3 2 4

simple.vector[2:4]
[1] 2 4 6
```

Babraham
Bioinformatics

Assigning names to vector slots

```
simple.vector
[1] 1 2 4 6 3

some.names
[1] "simon"  "laura"  "anne"   "jo"     "steven"

names(simple.vector)
NULL

names(simple.vector) <- some.names

simple.vector
 simon  laura   anne     jo steven
     1      2      4      6      3
```

Babraham Bioinformatics

Extracting by name

```
simple.vector
 simon  laura   anne      jo steven
     1      2      4       6      3

simple.vector["anne"]
 anne
    4

simple.vector[c("anne","simon","laura")]
 anne simon laura
    4     1     2
```

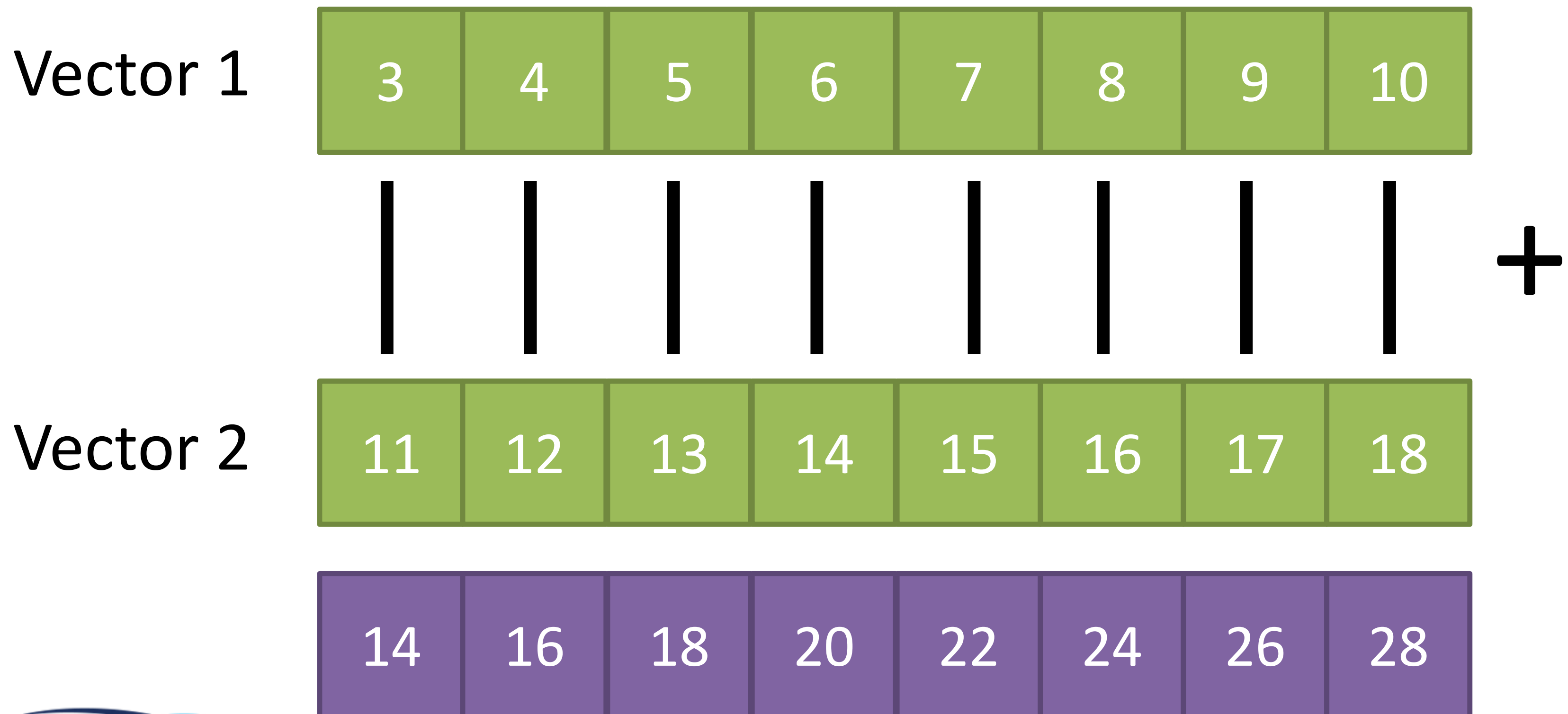
Babraham
Bioinformatics

Vectorised Operations

```
2+3
[1] 5

c(2,4) + c(3,5)
[1] 5 9

simple.vector
 simon  laura   anne     jo steven
     1      2      4      6      3

simple.vector * 100
 simon  laura   anne     jo steven
   100    200    400    600    300
```

Rules for vectorised operations

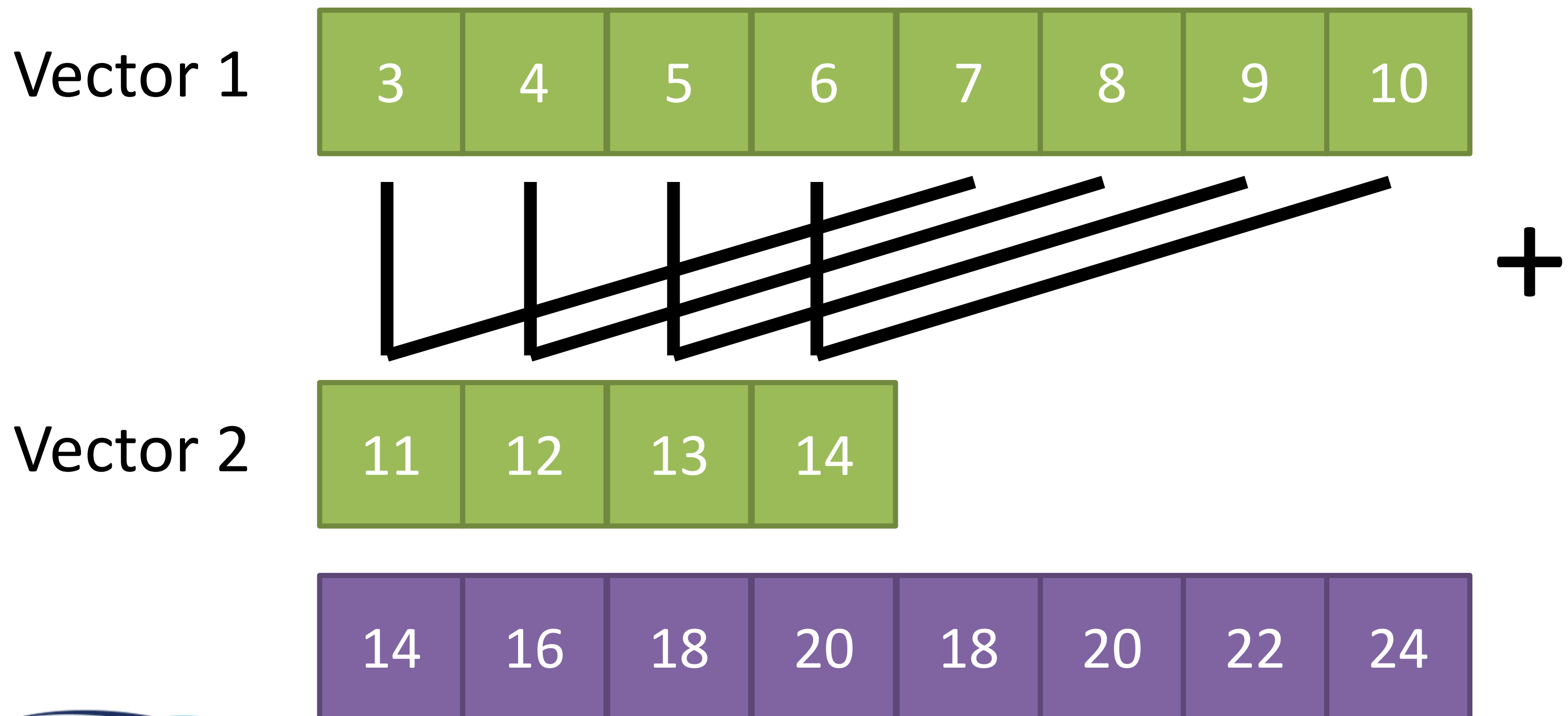
- Equivalent positions are matched



Babraham Bioinformatics

Rules for vectorised operations

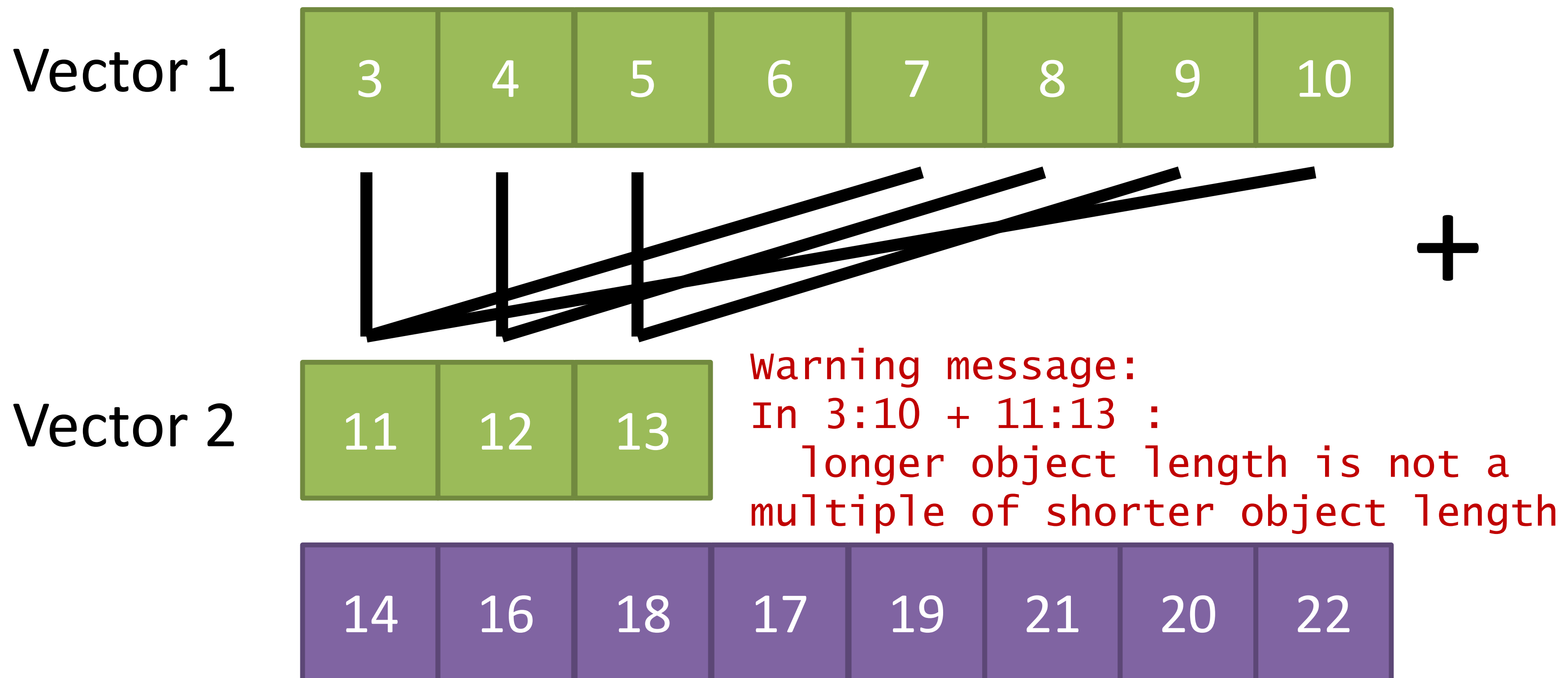
- Shorter vectors are recycled



Babraham Bioinformatics

Rules for vectorised operations

- Incomplete vectors generate a warning



Vectorised Operations

```
c(2,4) + c(3,5)
[1] 5 9

simple.vector
 simon  laura   anne     jo steven
     1      2      4      6      3

simple.vector * 100
 simon  laura   anne     jo steven
   100    200    400    600    300
```

Babraham Bioinformatics

Updating vectors

- Overwrite the existing vector

```
simple.vector
 simon  laura   anne     jo steven
     1      2      4      6      3

simple.vector[2:4] -> simple.vector

simple.vector
 laura  anne    jo
     2     4     6
```

Babraham Bioinformatics

Updating vectors

- Replace contents based on a selection

```
simple.vector
 simon  laura   anne     jo steven
     1      2      4      6      3

simple.vector[c("jo","laura")] <- c(200,500)

simple.vector
 simon  laura   anne     jo steven
     1    500      4    200      3
```

Babraham Bioinformatics

Exercise 2

R Data Structures

Babraham
Bioinformatics

Vector

- 1D Data Structure of fixed type

scores

	scores	
1	0.8	“bob”
2	1.2	“dave”
3	3.3	“mary”
4	1.8	“sue”
5	2.7	“alan”

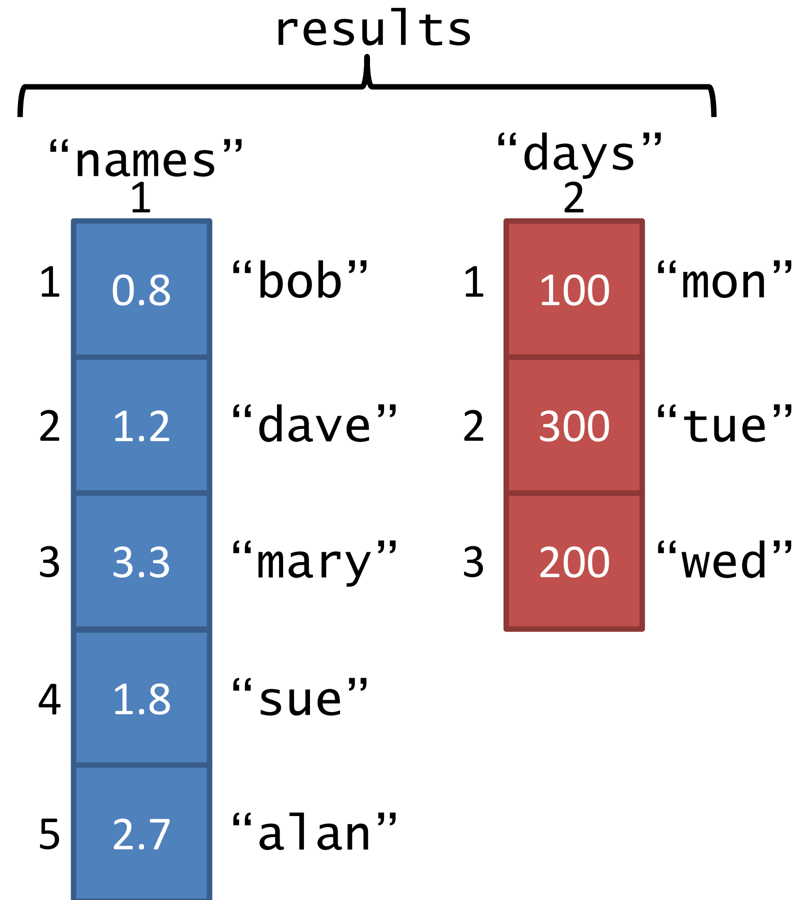
```
scores[2]
scores[c(2,4,3)]
scores[3:5]
scores[“mary”]
scores[c(“mary”,”sue”)]
```

Babraham Bioinformatics

List

- Collection of vectors

```
results[[1]]
results[["days"]]
results$days

results$days[2:3]
results[[1]]["sue"]
```



Babraham Bioinformatics

Data Frame

- Collection of vectors with same lengths

```
all.results[[1]]
all.results[["tue"]]
all.results$wed

all.results[5,2]
all.results[1:3,c(2,4)]
all.results[c("bob","dave"),]
all.results[,2:3]
```

all.results

		"mon" 1	"tue" 2	"wed" 3	"pass" 4
"bob"	1	0.8	0.9	0.8	T
"dave"	2	0.6	0.7	0.5	F
"mary"	3	0.2	0.3	0.3	F
"sue"	4	0.8	0.8	0.9	T
"alan"	5	0.6	1.0	0.9	T

Creating lists / data frames

- `list(vector1,vector2,vector3)`
- `data.frame(vector1,vector2,vector3)`

- `list(names=vector1,values=vector2)`
- `data.frame(names=vector1,values=vector2)`

- `names(my.list) <- c(“age”,“height”,“score”)`
- `colnames(my.df) <- c(“age”,“height”,“score”)`
- `rownames(my.df) <- c(“bob”,“dave”,“mary”,“sue”)`

Babraham Bioinformatics

Exercise 3

Spot the mistakes

```
vec1 <- c(31,47,15 52,13)
```

Error: unexpected numeric constant in "vec1 <- c(31,47,15 52“

```
vec2 <- c("Alfie","Bob","Chris",Dave,"Ed")
```

Error: object 'Dave' not found

```
vec3 <- (TRUE,TRUE,FALSE,  TRUE ,FALSE)
```

Error: unexpected ',' in "vec3 <- (TRUE,"

```
vec4 <- c[41, 67]
```

Error in c[41, 67] : object of type 'builtin' is not subsettable```

```
vec5 <- c("Alfie","Bob,"Chris","Dave")
```

Error: unexpected symbol in "vec5 <- c("Alfie","Bob,"Chris"

Babraham Bioinformatics

Spot the mistakes

```
my.vector(1:5)
    Error: could not find function "my.vector"
my.vector[2,3,4]
    Error in my.vector[2, 3, 4] : incorrect number of
    dimensions

my.list[2]
    [No error! Works – but don't do this]
my.data.frame[2:4]
    Error in `[.data.frame`(my.data.frame, 2:4) :
    undefined columns selected
nrow(my.data.frame)
[1] 10

my.data.frame[300,]

        a  b  c
    NA NA NA NA
```

Babraham Bioinformatics

Reading data from files

read.table {utils} R Documentation

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

Using read.table

- Only required parameter is the file name (path)
- Other parameters are optional

- You hardly ever call `read.table` directly
  - `read.delim` for tab delimited files
  - `read.csv` for comma separated value files

- The function returns a data frame - it *doesn't* save it. You need to do that

Babraham Bioinformatics

Specifying file paths

- You can use full file paths, but it's a pain

```
read.csv("O:/Training/Introduction to R/R_intro_data_files/neutrophils.csv")
```

- Easier to set the 'working directory' and then just provide a file name
  - `getwd()`
  - `setwd(`*path*`)`
  - Session > Set Working Directory > Choose Directory

- Use [Tab] to fill in file paths in the editor

Babraham
Bioinformatics

Being clear about names

- File names only matter when loading.
- After that the variable name is used

```
read.delim("data_file.txt") -> my.data

head(my.data)
```

Babraham Bioinformatics

Exercise 4

Logical Selection

```
> simple.vector
 simon  laura   anne      jo steven 
     1      2      4       6      3 
```

```
simple.vector[c(...)]
```

1. Numbers (index positions)
2. Text (names)
3. Logicals (TRUE/FALSE)

Logical Selection

```
simple.vector
 simon  laura   anne     jo steven
     1      2      4      6      3

c(TRUE,FALSE,FALSE,TRUE,FALSE)

simple.vector[c(TRUE,FALSE,FALSE,TRUE,FALSE)]

 simon     jo
     1      6
```

Babraham Bioinformatics

Logical Vectors are created by logical tests

```
simple.vector
 1       2       4       6       3

simple.vector > 3
 FALSE   FALSE    TRUE    TRUE   FALSE

simple.vector == 2
 FALSE    TRUE   FALSE   FALSE   FALSE

simple.vector <= 4
 TRUE    TRUE    TRUE   FALSE    TRUE
```

Babraham Bioinformatics

Combine the two concepts to make logical selections

```
simple.vector
 1      2      4      6      3

simple.vector > 3
 FALSE  FALSE   TRUE   TRUE  FALSE

simple.vector > 3 -> logical.result

simple.vector[logical.result]
 4    6

simple.vector[simple.vector > 3]
 4    6
```

Babraham Bioinformatics

Extension to data frames

- Select the people with heights over 170

```
trumpton
   LastName FirstName Age Weight Height
1      Hugh     Chris  26     90    175
2       Pew      Adam  32    102    183
3    Barney    Daniel  18     88    168
4    McGrew     Chris  48     97    155
5 Cuthbert      Carl   28     91    188
6    Dibble      Liam  35     94    145
7      Grub      Doug  31     89    164
```

Babraham Bioinformatics

3 Steps to Success!

1. Extract the column containing the data you want to filter against

2. Perform the logical test to get a logical vector

3. Use the logical vector to select the rows from the original data frame

Select people over 170 tall

1. Extract the column containing the data you want to filter against

```
trumpton
  LastName FirstName Age Weight Height
1     Hugh     Chris  26     90    175
2      Pew      Adam  32    102    183
3   Barney    Daniel  18     88    168
4   McGrew     Chris  48     97    155
5 Cuthbert      Carl  28     91    188
6   Dibble      Liam  35     94    145
7     Grub      Doug  31     89    164
```

```
trumpton$Height
```

Babraham Bioinformatics

Select people over 170 tall

2. Perform the logical test to get a logical vector

```
trumpton
  LastName FirstName Age Weight Height
1     Hugh     Chris  26     90    175
2      Pew      Adam  32    102    183
3   Barney    Daniel  18     88    168
4   McGrew     Chris  48     97    155
5 Cuthbert      Carl  28     91    188
6   Dibble      Liam  35     94    145
7     Grub      Doug  31     89    164
```

```
trumpton$Height > 170
```

Babraham Bioinformatics

Select people over 170 tall

3. Use the logical vector to select the rows from the original data frame

```
trumpton
  LastName FirstName Age Weight Height
1     Hugh     Chris  26     90    175
2      Pew      Adam  32    102    183
3   Barney    Daniel  18     88    168
4   McGrew     Chris  48     97    155
5 Cuthbert      Carl  28     91    188
6   Dibble      Liam  35     94    145
7     Grub      Doug  31     89    164
```

```
trumpton$Height > 170
trumpton[rows,columns]
trumpton[trumpton$Height > 170,]
```

Select people over 170 tall

```
trumpton[trumpton$Height > 170,]

  LastName FirstName Age Weight Height
1     Hugh     Chris  26     90    175
2      Pew      Adam  32    102    183
5 Cuthbert      Carl  28     91    188
```

Babraham Bioinformatics

It's not just selections...

- Sometimes you just want to know how many times something is true, rather than getting the values

- You can take the `sum()` of a logical vector to get the count of TRUE values

Babraham Bioinformatics

3.5 Steps to Success!

1. Extract the column containing the data you want to filter against

2. Perform the logical test to get a logical vector

3. Use the logical vector to select the rows from the original data frame

   3. Take the `sum()` of the logical vector to count hits

Babraham Bioinformatics

How many people are over 170 tall

```
sum(trumpton$Height > 170)

[1] 3
```

Using `subset` function for selections

- Select the people with heights over 170

```
subset(trumpton, Height>170)

  LastName FirstName Age Weight Height
1     Hugh     Chris  26     90    175
2      Pew      Adam  32    102    183
5 Cuthbert      Carl  28     91    188
```

Babraham Bioinformatics

Exercise 5

Babraham
Bioinformatics