



R for Data Science - 1

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Learning Objectives

By the end of this topic, you should be able to:

- Read the R syntax.
- Use the data structures/ objects of R.
- Deploy programming concepts in the syntax (for example, recursion, loops, variable assignment etc.).





**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

A Few Notes about R

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

R is a Statistical Programming Language

R compiles and runs on Windows, Mac OS X, and numerous UNIX platforms (such as Linux).

R is a “statistical programming” language providing an optimised environment and support for statistical computation and graphics.

It also allows integration with compiled code written in C, C++, Fortran, Java, etc., for computationally intensive tasks or for leveraging tools provided for other languages.

Download and install a copy of R from <http://cran.r-project.org>).

Why R?

For R

R is a highly customisable yet fully functional programming language.

R is powerful, expandable and is supported by an active data science/ math/ stats community.

R is free!

Amongst languages, R has the one of the most powerful graphics producing capabilities.

Against R

R is difficult to learn due to complex syntax.

R has many different data structures.

R has relatively weak text parsing ability.

R lacks organisation, which can breed bad coding habits in beginners.

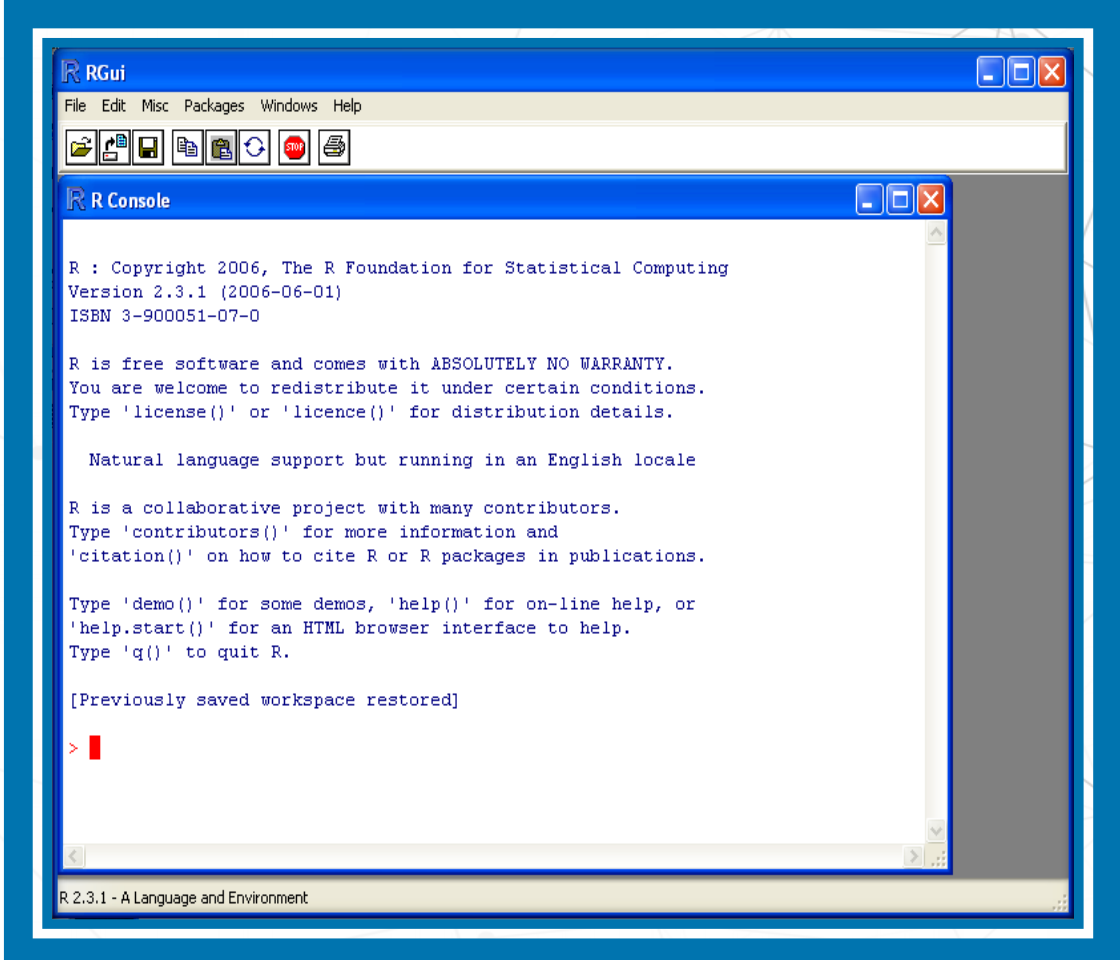
The GUI and Command Prompt

When running R, the first thing to observe is the Graphical User interface (GUI).

At the prompt (`>`), you can enter numbers and perform calculations e.g. `> 1 + 3`.

You also enter commands here although this is not a good place to develop code.

We will use an Integrated Development Environment (IDE) instead.



```
R RGui
File Edit Misc Packages Windows Help
R Console
R : Copyright 2006, The R Foundation for Statistical Computing
Version 2.3.1 (2006-06-01)
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> █
```

R 2.3.1 - A Language and Environment



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

RStudio

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

What is RStudio?

RStudio is not R.

It is a separate open-source project that brings many powerful coding tools together into an intuitive, easy-to-learn interface. Rstudio is an Integrated Development Environment (IDE).

It runs on all major platforms (Windows, Mac, Linux) and also through web browser (using the server installation).

Download and install a copy of Rstudio:
<http://www.rstudio.com/>

Why use an IDE?



An IDE facilitates code development.

It includes a console for issuing commands and a source-code editor with a rich set of keyboard shortcuts.

Other perks include automatic source-code formatting, assistance with parentheses, keyword highlighting, interfaces for compiling or running of software, project-management features, debugging assistance, and integration with report-writing tools.

RStudio can do all these things.

Layout of RStudio

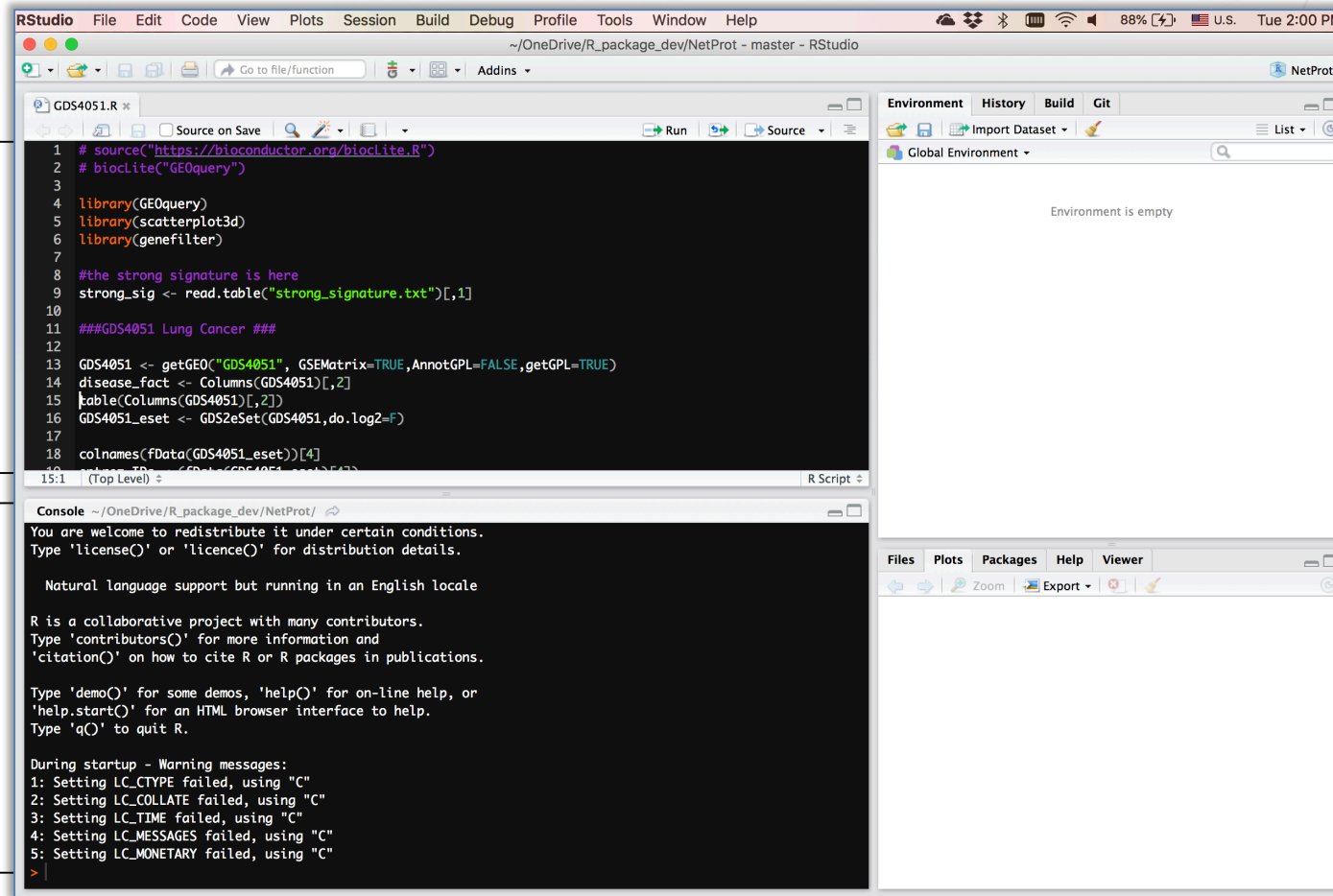
A tabbed source-code editor.

Console for interactive R sessions.

List of data objects already loaded into memory.

Notebooks Panel

The main components of RStudio are all nicely integrated into a four-panel layout that includes a console for interactive R sessions, a tabbed source-code editor to organise a project's files, panels with notebooks to organise less central components, and a list of data objects already loaded into memory.





**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

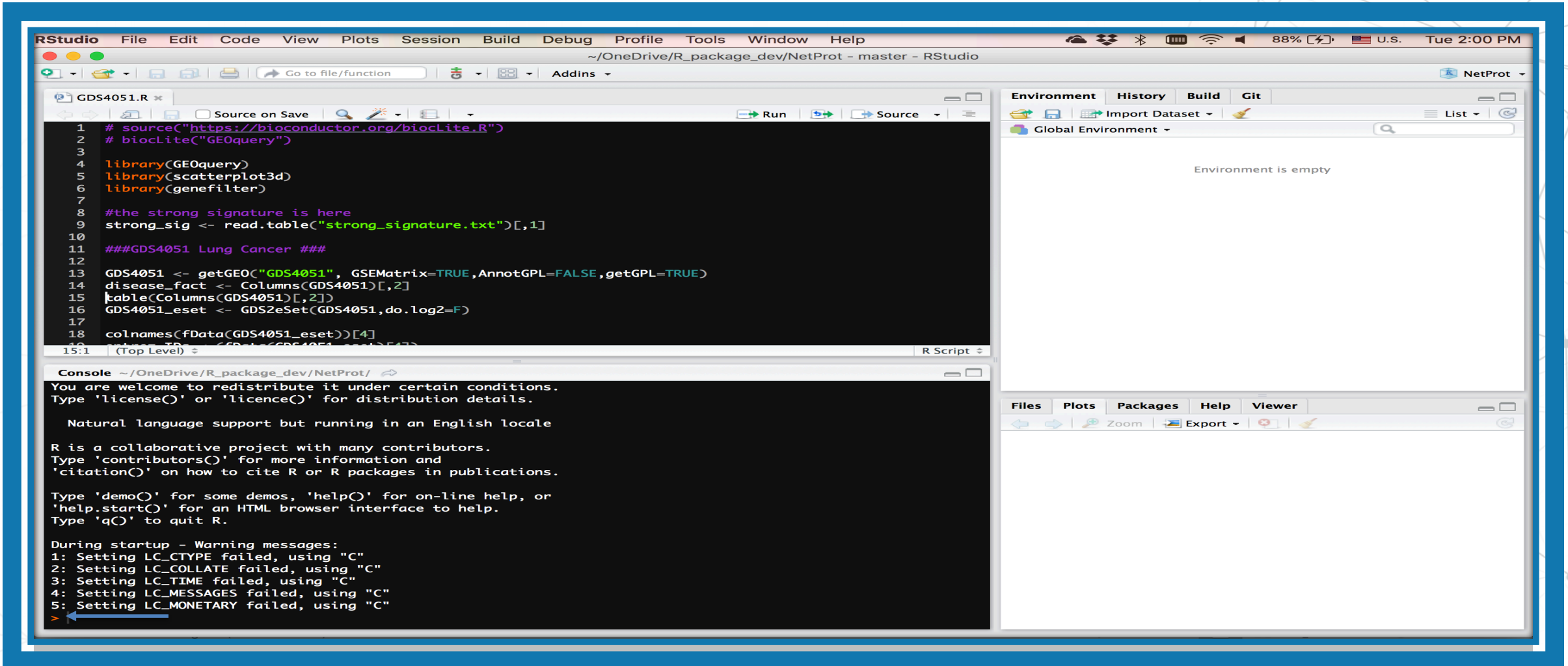
Entering Commands

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Entering Commands via the Console



The screenshot displays the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Window, and Help. The main editor window shows a script named 'GDS4051.R' with the following code:

```
1 # source("https://bioconductor.org/biocLite.R")
2 # biocLite("GEOquery")
3
4 library(GEOquery)
5 library(scatterplot3d)
6 library(genefilter)
7
8 #the strong signature is here
9 strong_sig <- read.table("strong_signature.txt")[,1]
10
11 ###GDS4051 Lung Cancer ###
12
13 GDS4051 <- getGEO("GDS4051", GSEMatrix=TRUE, AnnotGPL=FALSE, getGPL=TRUE)
14 disease_fact <- Columns(GDS4051)[,2]
15 table(Columns(GDS4051)[,2])
16 GDS4051_eset <- GDS2eSet(GDS4051, do.log2=F)
17
18 colnames(fData(GDS4051_eset))[4]
19
20
```

The console window at the bottom shows the R startup message:

```
Console ~/OneDrive/R_package_dev/NetProt/
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
>
```

The Environment pane on the right shows 'Global Environment' and 'Environment is empty'. The bottom toolbar includes Files, Plots, Packages, Help, and Viewer.

Console for interactive R sessions.



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Basic Operations and Assignment

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Operations on the Console

Console ~/ ↵

```
> 2 + 2  
[1] 4  
> 2 +  
+ |
```

Console ~/ ↵

```
+ <  
[1] 4  
> 2 _  
Error: unexpected input in "2 _"  
>
```

Assignment

- To assign a value to a variable:
 - Type “<-” which means ←.
 - The equals (=) sign can also be used.
 - For example, `x = 1` or `x <- 1` means the same thing in R.
-
- However, seasoned R programmers prefer to use `<-`.

Basic Arithmetic Operations and Value Assignment

Arithmetic operations $+$, $-$, $*$, $/$ and $^$ are the standard arithmetic operators.

| Operator | Meaning |
|----------|--|
| $+$ | Plus/Summation |
| $-$ | Minus/Subtraction |
| $*$ | Times/Multiplication |
| $/$ | Division (where a/b means a divide over b) |
| $^$ | Power of where a^b means a to the power of b or simply a^b |



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Writing and Using Functions in R

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

What is a Function?

A function is a well-defined autonomous piece of code.

It can contain both data and methods.

It is conceptually related to OOP. But it differs in the sense that functions are not initiated from a constructor class (no inheritance).

R has many built in functions that can conveniently be called upon.

Invoking R Functions

R functions are invoked by their names, followed by a parenthesis, and include zero or more arguments.

The below example applies the *combine* function *c()* to merge three numeric values into a vector.

`> c(1,2,3)` will return 1 2 3.

Similarly, using the *sum()* function as *sum(1,2,3)* or *sum(c(1,2,3))* will return 6.

Creating Your Own Function

R has many functions, which for most purposes, are sufficient to meet needs.

There are occasions however, when you have to create your own function.

This can be expressed in R as follows:

```
myfunction <- function(arg1, arg2, ...) {  
  statements  
  return(object)  
}
```

Creating Your Own Function

Using the function command to create a new function “my_first_function()”.

```
3 #my_first_function is the name of this function
4 my_first_function <- function(x) #<- input is x
5 {
6   y <- sum(x) # first statement instructs what will be done in this function
7   return(y) #instructs this function to return the variable y as output
8 }
9
```

Console shows no error.

```
> #my_first_function is the name of this function
> my_first_function <- function(x) #<- input is x
+ {
+   y <- sum(x) # first statement instructs what will be done in this function
+   return(y) #instructs this function to return the variable y as output
+   }
>
```

Creating Your Own Function

```
3 #my_first_function is the name of this function
4 my_first_function <- function(x) #<- input is x
5 {
6   y <- sum(x) # first statement instructs what will be done in this function
7   return(y) #instructs this function to return the variable y as output
8 }
9
10 my_first_function(c(1,2,3))
```

```
> my_first_function(c(1,2,3))
[1] 6
```

This function as you can see, does not actually do much. It is basically calling the *sum()* function within itself and returning the output from *sum()*. However, the ability to combine and call functions within functions can be very powerful and neater in terms of organisation. It is also a useful option when writing complex code requiring repetitive components.



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Comments in R

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Comments

Code

```
3 #my_first_function is the name of this function
4 my_first_function <- function(x) #<- input is x
5 {
6   y <- sum(x) # first statement instructs what will be done in this function
7   return(y) #instructs this function to return the variable y as output
8 }
9
```

Console

```
> #my_first_function is the name of this function
> my_first_function <- function(x) #<- input is x
+ {
+   y <- sum(x) # first statement instructs what will be done in this function
+   return(y) #instructs this function to return the variable y as output
+ }
> |
```

- All the text after the pound sign "#" within the same line is considered a comment.
- A comment will not be run as part of the program. It tells you what each part of the program does. Appropriate use of the pound sign is really important for documentation and forms part of good coding practice.
- If you remove the pound sign from the code, errors will occur.



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Extensions and Getting Help

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Extension Package

R comes prebuilt with many useful functions but those aren't always enough.

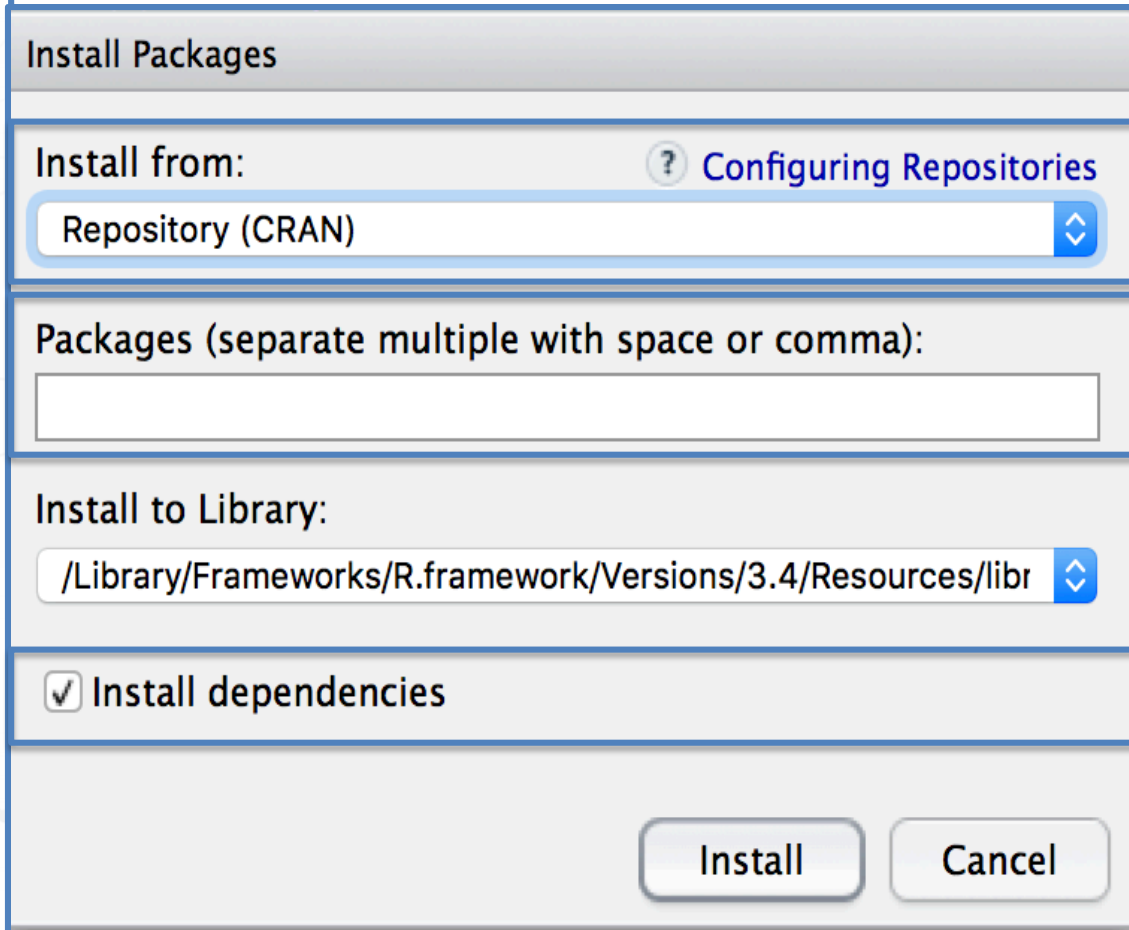
You don't have to write your own functions if someone else has already written them.

To install an extension package, use the **install.packages()** function on the console.

If you know the package name already, use **install.packages(<package name>)**.

Extension Package in RStudio

Go to **Tools** → **Install Packages**, you will see several options.



The screenshot shows the 'Install Packages' dialog box in RStudio. It has a title bar 'Install Packages'. Below the title bar, there are several sections: 'Install from:' with a dropdown menu set to 'Repository (CRAN)' and a link to 'Configuring Repositories'; 'Packages (separate multiple with space or comma):' with an empty text input field; 'Install to Library:' with a dropdown menu set to '/Library/Frameworks/R.framework/Versions/3.4/Resources/libr'; and a checked checkbox for 'Install dependencies'. At the bottom, there are two buttons: 'Install' and 'Cancel'.

The **Install from** drop down provides choices on whether you want to scan for packages available in the online repository Comprehensive R Archive Network (CRAN), or if you want to install a locally downloaded package.

Multiple packages can also be installed simultaneously. Some packages require other packages to work. They are thus dependent on these.

Selecting **Install dependencies** auto installs these other packages without you specifying them explicitly.

CRAN

R is extensible via packages to supplement the base distribution.

This is supported via a worldwide repository system, the CRAN available on the below website: <http://cran.r-project.org>

As of 2011, there are more than 3,000 such packages hosted on CRAN.

Getting Help

- R has a very good built-in help system.
- If you know which function you want help with simply use **?<function name>**.
- For example, typing **?hist** will produce the following in the notebook panel.

```
hist {graphics}
```

R Documentation

Histograms

Description

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of [class](#) "histogram" is plotted by [plot.histogram](#), before it is returned.

Usage

```
hist(x, ...)
```

```
## Default S3 method:
```

```
hist(x, breaks = "Sturges",
```

Getting Help

- If you are looking for something more general, or you do not know the name of the exact function. You may use the **help.search()** function.
- For example, typing **help.search("histogram")** will produce the below shown results. If not, Google and programming forums are always your best friends.

Search Results

Help pages:

| | |
|--|--|
| KernSmooth::dpih | Select a Histogram Bin Width |
| MASS::hist.scott | Plot a Histogram with Automatic Bin Width Selection |
| MASS::ldahist | Histograms or Density Plots of Multiple Groups |
| MASS::truehist | Plot a Histogram |
| caret::densityplot.rfe | Lattice functions for plotting resampling results of recursive feature selection |
| caret::histogram.train | Lattice functions for plotting resampling results |
| ggplot2::geom_freqpoly | Histograms and frequency polygons |
| grDevices::nclass.Sturges | Compute the Number of Classes for a Histogram |
| graphics::hist.POSIXt | Histogram of a Date or Date-Time Object |
| graphics::hist | Histograms |
| graphics::plot.histogram | Plot Histograms |
| lattice::histogram | Histograms and Kernel Density Plots |
| lattice::panel.histogram | Default Panel Function for histogram |
| lattice::prepanel.default.bwplot | Default Prepanel Functions |
| sfsmisc::histBxp | Plot a Histogram and a Boxplot |



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Types in R

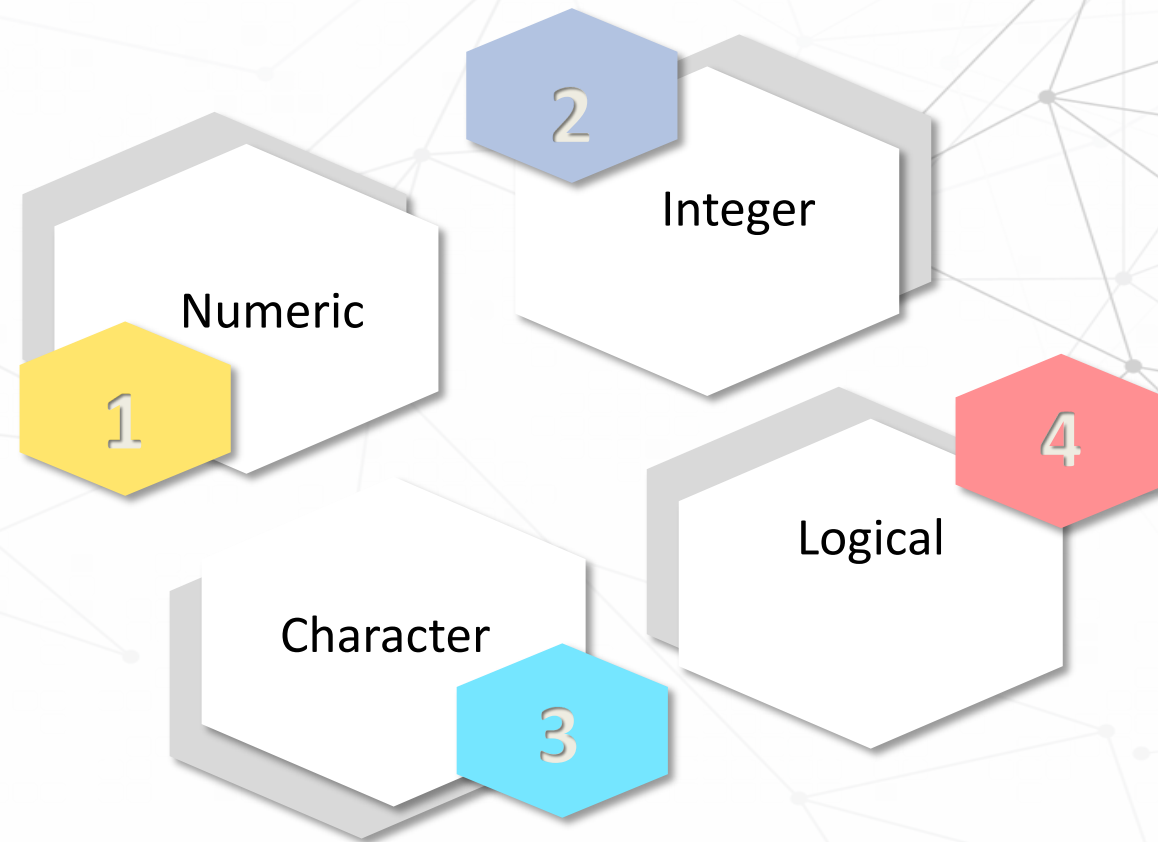
BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Data Types

R needs to know what kind of data we are dealing with. And this in turn, dictates what functions and methods are available. The data can be of the following types:

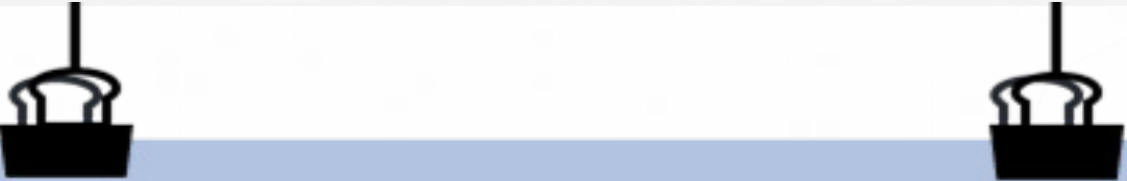


Numeric

Decimal values are called numeric in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

```
x = 10.5    # assign a decimal value
x           # print the value of x
x
[1] 10.5
class(x)    #what is the class of data x belongs to?
[1] "numeric"
```

Integer



An integer is a whole number, but you cannot invoke it simply by assigning a whole number to a variable, e.g. `Y <- 1`.

```
is.integer(Y)  
[1] False #it is considered a numeric
```

Instead we use the `as.integer` function e.g. `y = as.integer(3); class(y)`. Assigning a string variable, e.g. running `as.integer("Wilson")` on the console will return the message ...

Complex

A complex type in R is an imaginary number, and we use the imaginary value i to assign this.

```
> z = 1 + 2i # create a complex number
> z          # print the value of z
[1] 1+2i
> class(z)   # print the class name of z
[1] "complex"
```

You will seldom use this, so we will skip ahead.

Logical

A logical value (True or False) is generated via comparisons between variables.

```
> x = 1; y = 2 # sample values
> z = x > y    # is x larger than y?
> z           # print the logical value
[1] FALSE
> class(z)    # print the class name of z
[1] "logical"
```


Character

Character object is used to store string values (e.g. "Apple") in R.

It can also be used to convert numeric objects into strings.

```
> x = as.character(3.14)
> x           # print the character string
[1] "3.14"
> class(x)    # print the class name of x
[1] "character"
```

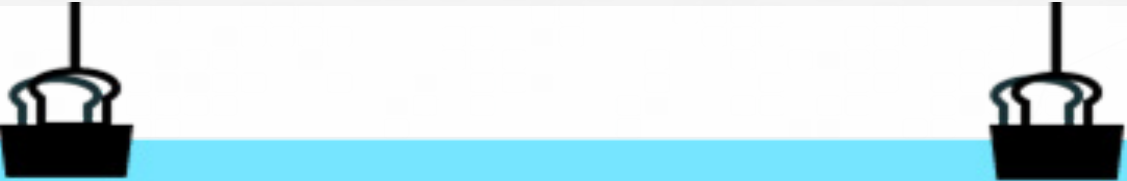
Character



To extract a substring, we apply the **substr** function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

```
> substr("Mary has a little lamb.", start=3,  
stop=12)  
[1] "ry has a l"
```

Character



To replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

```
> sub("little", "big", "Mary has a little lamb.")  
[1] "Mary has a big lamb."
```



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Structures in R

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Data Structures in R

Data types can be assembled into larger and more complex entities called data structures. R offers a wide variety of data structures for satisfying different task requirements.



Vectors

Lists

Matrices

Data Frames

Factors

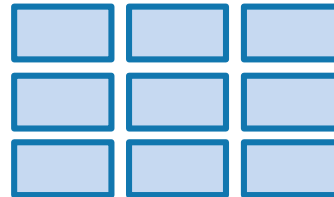
Data Structures in R

Vector



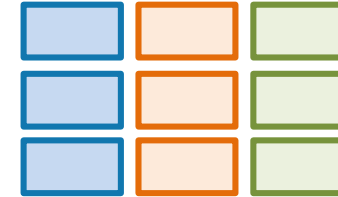
- 1 column or row of data
- 1 type (numeric or text)

Matrix



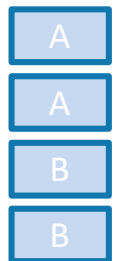
- Multiple columns and/or rows of data
- 1 type (numeric or text)

Data Frame



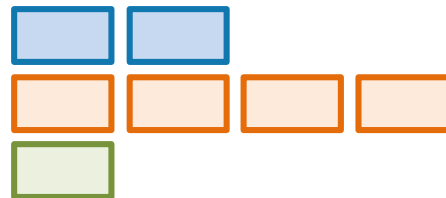
- Multiple columns and/or rows of data
- Multiple types

Factor



- It is 1 column or row
- Contains “level” data which describes “levels” of classification e.g. class label A or B

List



- A collection of entities with different lengths
- Multiple types
- Multiple data structures (vectors, matrices and data frames)



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Structures in R: Vectors

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Vectors

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called components or members.

Vectors may be created using the `c()` function:

```
1 c("Hola", "Ciao", "Hello", "Bonjour") # character vector
2 c(0.99, 2.4, 1.4, 5.9) # numeric vector
3 c(1L, 2L, 3L, 4L) # integer vector
4 c(TRUE, TRUE, FALSE, TRUE) # logical vector
```

Check the data type using the `class()` function:

```
1 class(c("Hola", "Ciao", "Hello", "Bonjour"))
2 class(c(0.99, 2.4, 1.4, 5.9))
3 class(c(1L, 2L, 3L, 4L))
4 class(c(TRUE, TRUE, FALSE, TRUE))
```

This will return “character”, “numeric”, “integer” and “logical”.

Vector



- 1 column or row of data
- 1 type (numeric or text)

Vectors

Other ways of creating vectors

Using seq()

```
1 seq(from = 1, to = 4, by = 1)
2 seq(from=1, to=4) # by=1 is default
3 seq(1, 4) # arguments in R can be matched by position
4 1:4 # common operations in R have shortcuts
```

Using rep()

```
1 rep(x = "a", times = 4) # replicate "a" four times
2 rep("a", 4) # same as above
3 rep(c("a", "b"), times = 2) # same but for a vector
4 rep(c("a", "b"), each = 2) # element-by-element
```

Vector Index

We retrieve values in a vector by declaring an index inside a single square bracket "[]" operator.

```
S = c("aa", "bb", "cc", "dd", "ee")
```

S =

| | | | | |
|----|----|----|----|----|
| aa | bb | cc | dd | ee |
|----|----|----|----|----|

The location of each element is marked by a position index.

Position 1 2 3 4 5

S =

| | | | | |
|----|----|----|----|----|
| aa | bb | cc | dd | ee |
|----|----|----|----|----|

Position 1 2 3 4 5

S[3] =

| | | | | |
|----|----|----|----|----|
| aa | bb | cc | dd | ee |
|----|----|----|----|----|

```
S[3] = c("cc")
```

Negative Vector Index

If the index is negative, it would strip the member whose position has the same absolute value as the negative index.

| Position | 1 | 2 | 3 | 4 | 5 |
|----------|----|----|----|----|----|
| S = | aa | bb | cc | dd | ee |

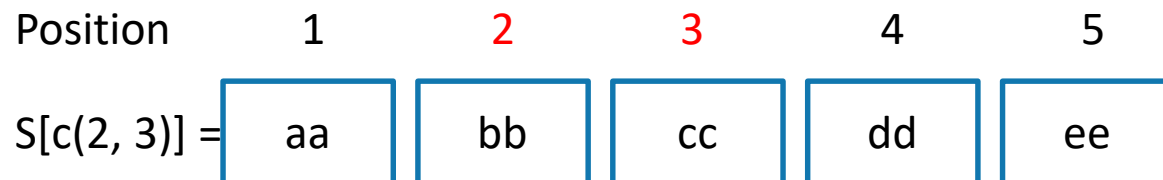
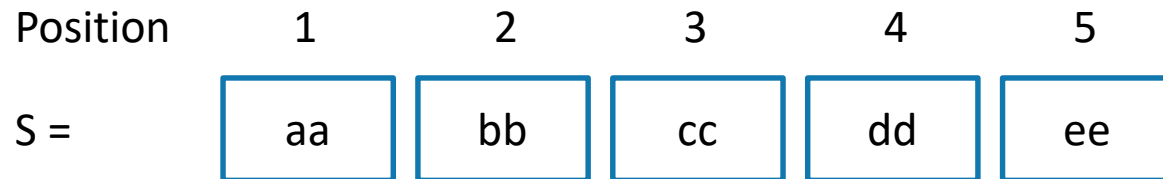
| Position | 1 | 2 | 3 | 4 | 5 |
|----------|----|----|--------------|----|----|
| S[-3] = | aa | bb | cc | dd | ee |

S[-3] = c("aa", "bb", "dd", "ee")

If an index is out-of-range, a missing value will be reported via the symbol NA. For example, S[10] will return NA.

Vector Slicing

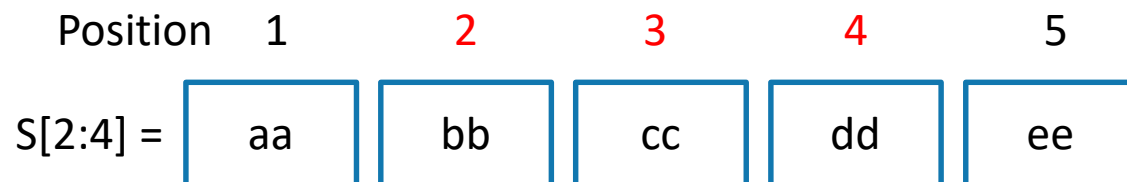
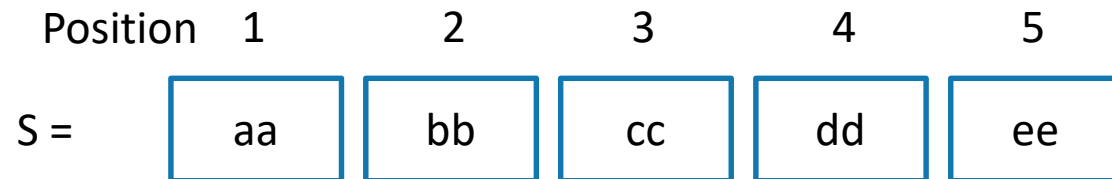
A new vector, S , can be sliced from a given vector with a numeric index vector, which consists of member positions of the original vector to be retrieved.



$S[c(2, 3)] = c("bb", "cc")$

Vector Slicing

Or more simply, we can simply supply a range index, for example, `S[Start:End]`.



`S[2:4] = c("bb", "cc", "dd")`

Vector Subsetting

Vectors can be subsetted by specifying a condition. Let's create a vector with values from 1 to 5; $S = 1:5$.

| | | | | | |
|----------|---|---|---|---|---|
| Position | 1 | 2 | 3 | 4 | 5 |
| S = | 1 | 2 | 3 | 4 | 5 |

We now specify a condition on S, $S[S < 3]$.

| | | | | | |
|----------|---|---|---|---|---|
| Position | 1 | 2 | 3 | 4 | 5 |
| S = | 1 | 2 | 3 | 4 | 5 |

$$S[S < 3] = c(1,2)$$

Performing Arithmetic on Vectors

Arithmetic operations of vectors are performed member-by-member.

$$a = c(1, 3, 5, 7) \quad b = c(1, 2, 4, 8)$$

| Position | 1 | 2 | 3 | 4 |
|----------|---|----|----|----|
| a = | 1 | 3 | 5 | 7 |
| b = | 1 | 2 | 4 | 8 |
| a + b = | 2 | 5 | 9 | 15 |
| a * 5 = | 5 | 15 | 25 | 35 |

Named Vectors

Members in a vector can have names. This is useful when you want to access a member by its name rather than by its position count.

```
V = c("Mary", "Sue")
```

```
Position      1      2
```

```
V = 

|      |     |
|------|-----|
| Mary | Sue |
|------|-----|


```

We now name the first member as First, and the second as Last.

```
names(V) = c("First", "Last")
```

```
Position      1      2
```

```
Name         "First"  "Last"
```

```
V = 

|      |     |
|------|-----|
| Mary | Sue |
|------|-----|


```

Named Vectors

Members in a vector can have names. This is useful when you want to access a member by its name rather than by its position count.

```
names(V) = c("First", "Last")
```

```
Position      1      2
```

```
Name         "First" "Last"
```

```
V = 

|      |     |
|------|-----|
| Mary | Sue |
|------|-----|


```

Instead of using numerical index, we can now retrieve the first member by its name.

```
Position      1      2
```

```
Name         "First" "Last"
```

```
V["First"] = 

|      |     |
|------|-----|
| Mary | Sue |
|------|-----|


```

```
V["First"] = "Mary"
```

Named Vectors

We can even reverse the order of V with a character string index vector containing the names.

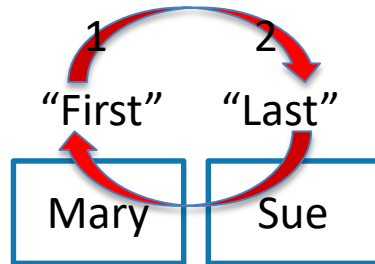
```
V[c("Last", "First")]
```

```
names(V) = c("First", "Last")
```

Position

Name

```
V [c("Last", "First")] =
```



```
V [c("Last", "First")] =
```





**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Structures in R: Matrices

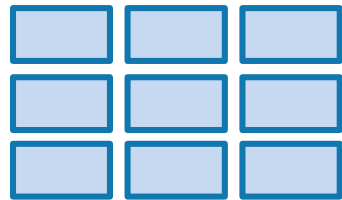
BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Matrices

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.



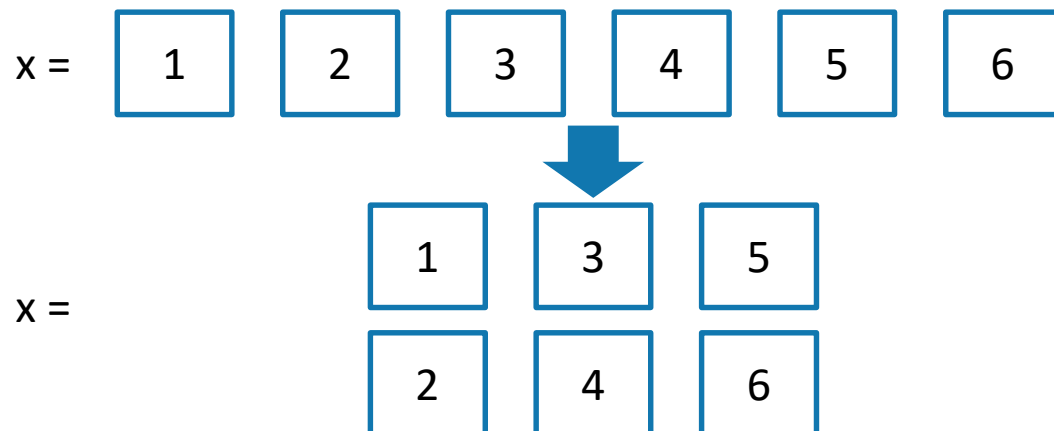
- Multiple columns and/or rows of data
- 1 type (numeric or text)

Matrix

Building Matrices

We reproduce a memory representation of the matrix in R with the `matrix()` function. The data elements must be of the same data type. There are several ways of using the `matrix()` function. A not so common way:

```
1 x <- 1:6 # take a vector
2 dim(x) # vector do not have dimension attribute
3 dim(x) <- c(2, 3) # impose a 2x3 dimesion (2 rows, 3 columns)
4 class(x) # here it is a matrix!
5 x
```



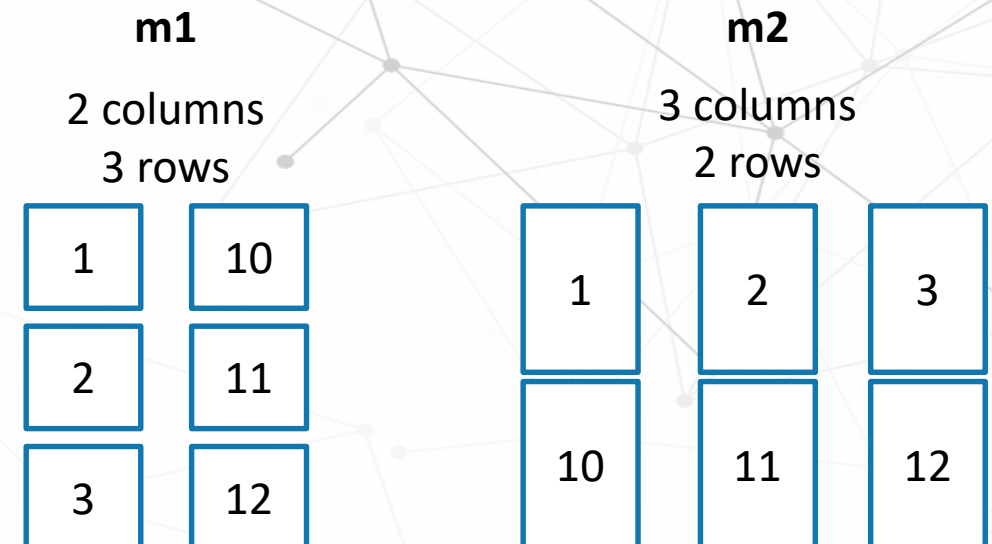
Matrix

The earlier expression can be made more elegant by writing it as one line.

```
1 m <- matrix(data = 1:6, nrow = 2, ncol = 3)
2 class(m)
3 dim(m)
```

We can also combine 2 vectors of similar length using row bind (rbind) or column bind (cbind).

```
1 x <- 1:3
2 y <- 10:12
3 m1 <- cbind(x,y)
4 m2 <- rbind(x,y)
5 class(m1)
6 class(m2)
```



Accessing Parts of Matrices

You may access individual elements by $A[x, y]$, where x is the row number and y is the column number.

| | | | | | |
|---------|----|----|----------|----|----|
| A = | | | A[,1:3]= | | |
| 1 | 2 | 3 | 1 | 2 | 3 |
| 10 | 11 | 12 | 10 | 11 | 12 |
| A[2,] = | | | A[,3]= | | |
| 10 | 11 | 12 | 3 | | |
| | | | 12 | | |



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Structures in R: Factors

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Factors

Factors are used to describe entities (samples) that can take on a class label (a category) e.g. disease or normal, rich or poor.

Unlike vectors, factors can take on only a finite set of values (levels), as many categories as there are e.g. rich and poor (number of levels = 2); good, moderate, excellent (number of levels = 3).

Factors are initiated using the `factor()` function.

```
1 f <- factor( c("f", "m", "m", "f", "f") )  
2 class(f)
```

Factor



- It is 1 column or row
- Contains “level” data which describes “levels” of classification e.g. class label A or B

Factors and Levels

```
1 f <- factor( c("f", "m", "m", "f", "f") )  
2 class(f)
```

Factors have a levels attribute listing its unique categories. Access levels attribute with levels() function.

In which case we will get "f" "m".

Changing Level Ordering

Consider the following factor, fo:

```
1 fo <- factor( c("low", "med", "low", "high"), ordered = TRUE)
```

Factor levels follow numerical or alphabetical ordering. So running `levels(fo)` will naturally return a vector as “high”, “low”, “med”, which doesn’t really make sense to us. We can fix this by specifying the order ourselves.

```
1 levels(fo) <- c("low", "med", "high") # re-order
```

Subsetting Data Using Factors

Expression Data Matrix

Sample

| | 1 | 2 | 3 |
|-----|---|---|---|
| x = | 2 | 3 | 5 |
| | 2 | 4 | 6 |
| | 2 | 4 | 6 |

Factor

```
F = factor(c("A","A","B"))
```

| | | |
|---|---|---|
| A | A | B |
|---|---|---|

`F=="A"` gives us a logical vector
TRUE TRUE FALSE

We may use this expression to
extract from X all the samples
corresponding to class A.

Sample

Sample

| | 1 | 2 |
|--------------|---|---|
| X[,F=="A"] = | 2 | 3 |
| | 2 | 4 |
| | 2 | 4 |



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Structures in R: Data Frames

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Data Frame

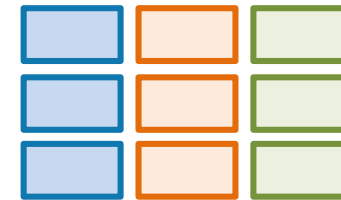
A data frame is used for storing data tables. It is less strict than a matrix, allowing different data types to be incorporated. It is a collection of vectors and/or factors all having the same length. A data frame generally has column names and row names attributes. You instantiate a data.frame with function `data.frame()`.

```
1 df <- data.frame( x = 1:3, y = c("a", "b", "c"),  
2 f = factor( c("m", "f", "m") ) )  
3 class(df)
```

| names | x | y | f |
|-------|---|---|---|
| df = | 1 | a | m |
| | 2 | b | f |
| | 3 | c | m |

x is numeric
y is character
f is factor

Data Frame



- Multiple columns and/or rows of data
- Multiple types

Although more often we auto-create data.frame by reading some data from a file using the `read.table()` function

Exploring Data Frames

R provides some example data that can be called using the `data()` function.

```
1 data("iris") # load the example data in the workspace
```

The iris' data.frame which gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris.

To explore the data frame, the following functions are useful:

```
1 str(iris) # returns a compact summary of R objects
2 summary(iris) # few statistics for each variable
3 head(iris, n = 20) # visualize first 20 observations
4 tail(iris) # last 6 observations
```

Subsetting Data Frames

Like matrices, the $[i,j]$ -index notation is valid also for data.frames.

| names | x | y | f |
|-------|---|---|---|
| df = | 1 | a | m |
| | 2 | b | f |
| | 3 | c | m |

df[,1] =

| |
|---|
| 1 |
| 2 |
| 3 |

df[2,1] =

| |
|---|
| b |
|---|

Subsetting Data Frames

Alternatively, we may also access parts of the data frame via name.

| names | x | y | f |
|-------|---|---|---|
| df = | 1 | a | m |
| | 2 | b | f |
| | 3 | c | m |

Using the \$ notation

| | |
|---------|---|
| df\$y = | a |
| | b |
| | c |

Quoting the name in the jth slot

| | |
|------------|---|
| df[,"f"] = | m |
| | f |
| | m |

| | | |
|-------------------|---|---|
| df[,c("x","f")] = | 1 | m |
| | 2 | f |
| | 3 | m |



**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Data Structures in R: Lists

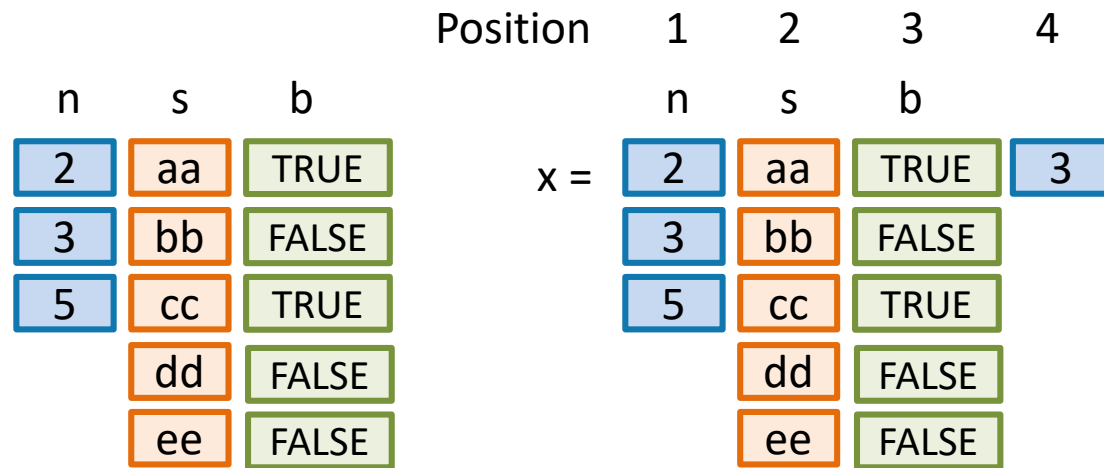
BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Lists

A list is a generic vector that can contain multiple data types. Unlike a data frame, it can contain multiple data structures of different dimensions. A list is instantiated using the `list()` function.



numeric character logical

List Slicing

We retrieve a list slice with the single square bracket "[" operator. The following is a slice containing the second member of x, which is a copy of s.

| Position | 1 | 2 | 3 | 4 |
|----------|---|----|-------|---|
| | n | s | b | |
| x[2] = | 2 | aa | TRUE | 3 |
| | 3 | bb | FALSE | |
| | 5 | cc | TRUE | |
| | | dd | FALSE | |
| | | ee | FALSE | |

| | |
|--------|----|
| x[2] = | aa |
| | bb |
| | cc |
| | dd |
| | ee |

List Slicing

We may access multiple elements of a list by specifying a vector of position indices.

| Position | 1 | 2 | 3 | 4 |
|---------------|---|----|-------|---|
| | n | s | b | |
| $x[c(2,4)] =$ | 2 | aa | TRUE | 3 |
| | 3 | bb | FALSE | |
| | 5 | cc | TRUE | |
| | | dd | FALSE | |
| | | ee | FALSE | |

| | | |
|---------------|----|---|
| $x[c(2,4)] =$ | aa | 3 |
| | bb | |
| | cc | |
| | dd | |
| | ee | |

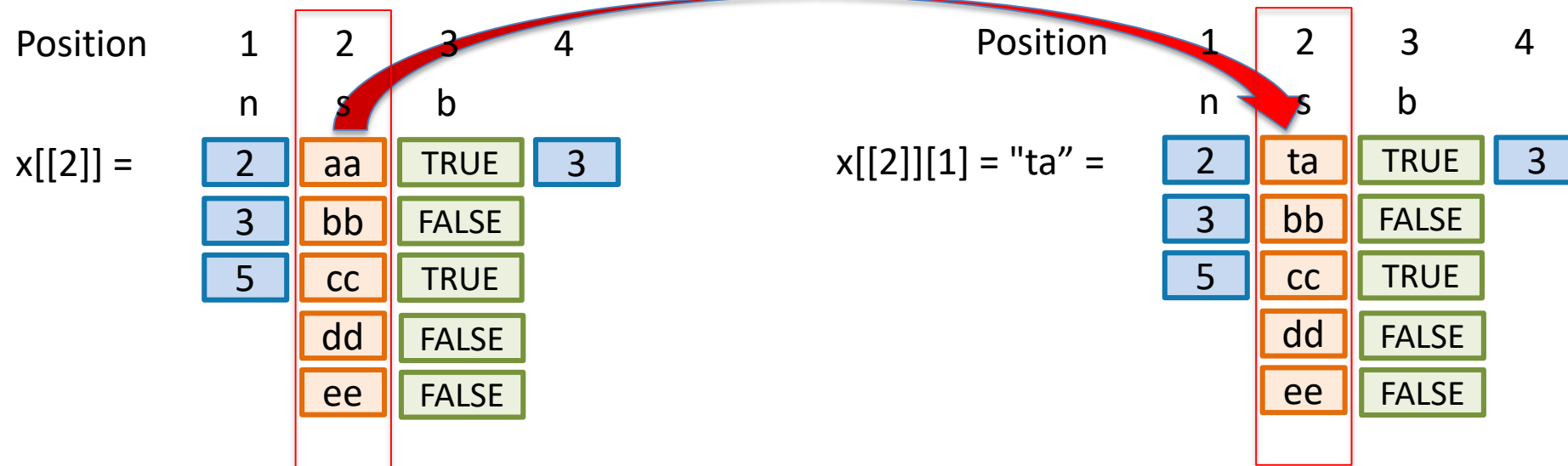
List Member Reference

List entities can be accessed via double brackets `[[]]`. This is a member reference, and allows us to access a part of the list instead of subsetting it as a separate entity

| Position | 1 | 2 | 3 | 4 |
|-----------------------|---|----|-------|---|
| | n | s | b | |
| <code>x[[2]] =</code> | 2 | aa | TRUE | 3 |
| | 3 | bb | FALSE | |
| | 5 | cc | TRUE | |
| | | dd | FALSE | |
| | | ee | FALSE | |

List Member Reference

The use of `[[]]` allows us to change values inside `x`.

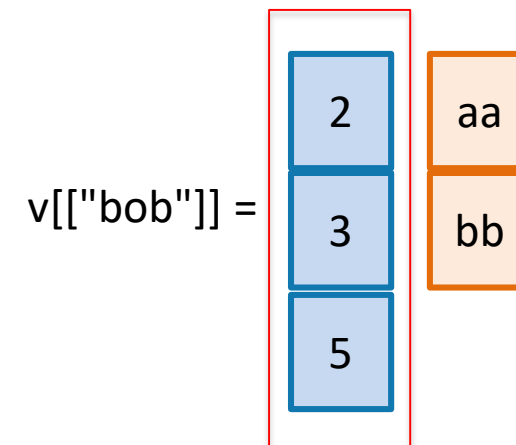
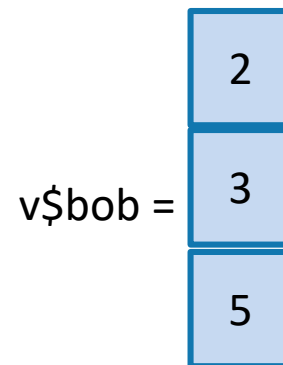
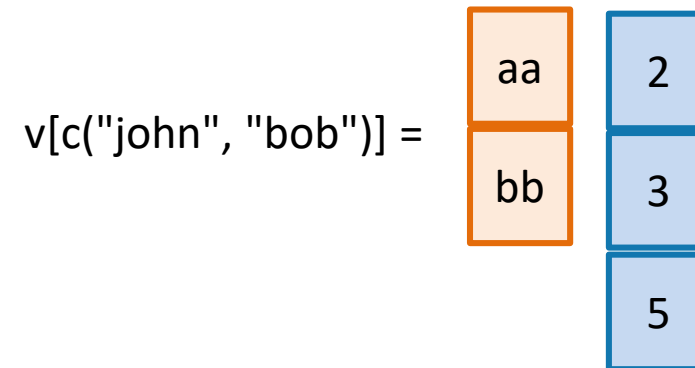
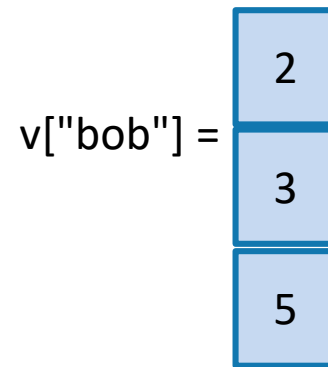
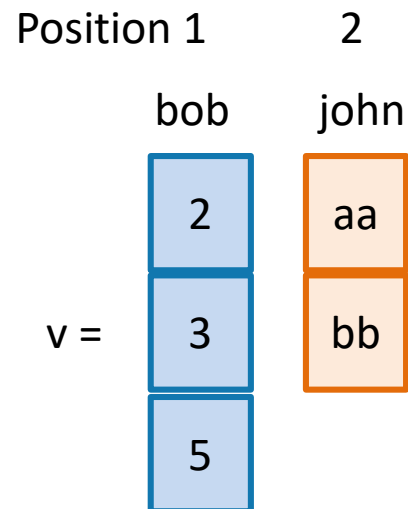


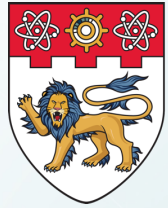
We access list element 2, at its first position. And changed its value

List Member Names

We can assign names to list members, and reference them by names instead of numeric indexes.

```
v = list(bob=c(2, 3, 5), john=c("aa", "bb"))
```





**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Summary

BS3033 Data Science for Biologists

Dr Wilson Goh

School of Biological Sciences

Key Takeaways from this Topic

1. The R language syntax is quite flexible and allows for various approaches for tackling the same problem. This flexibility is extremely powerful, but can also make R difficult to learn. A key feature of R is it allows you to get quite far with interactive programming, as you executing simple expressions line-by-line from a script into the R console. This interactive approach to programming allows development on the fly. But can also breed bad habits in new programmers.

2. R offers a wide variety of data structures for satisfying different task requirements. The basic data structures in R are vectors, lists, matrices, data frames and factors.

