BIOS 721|TOPIC 1: OBJECTS IN R – PART 1

Outline

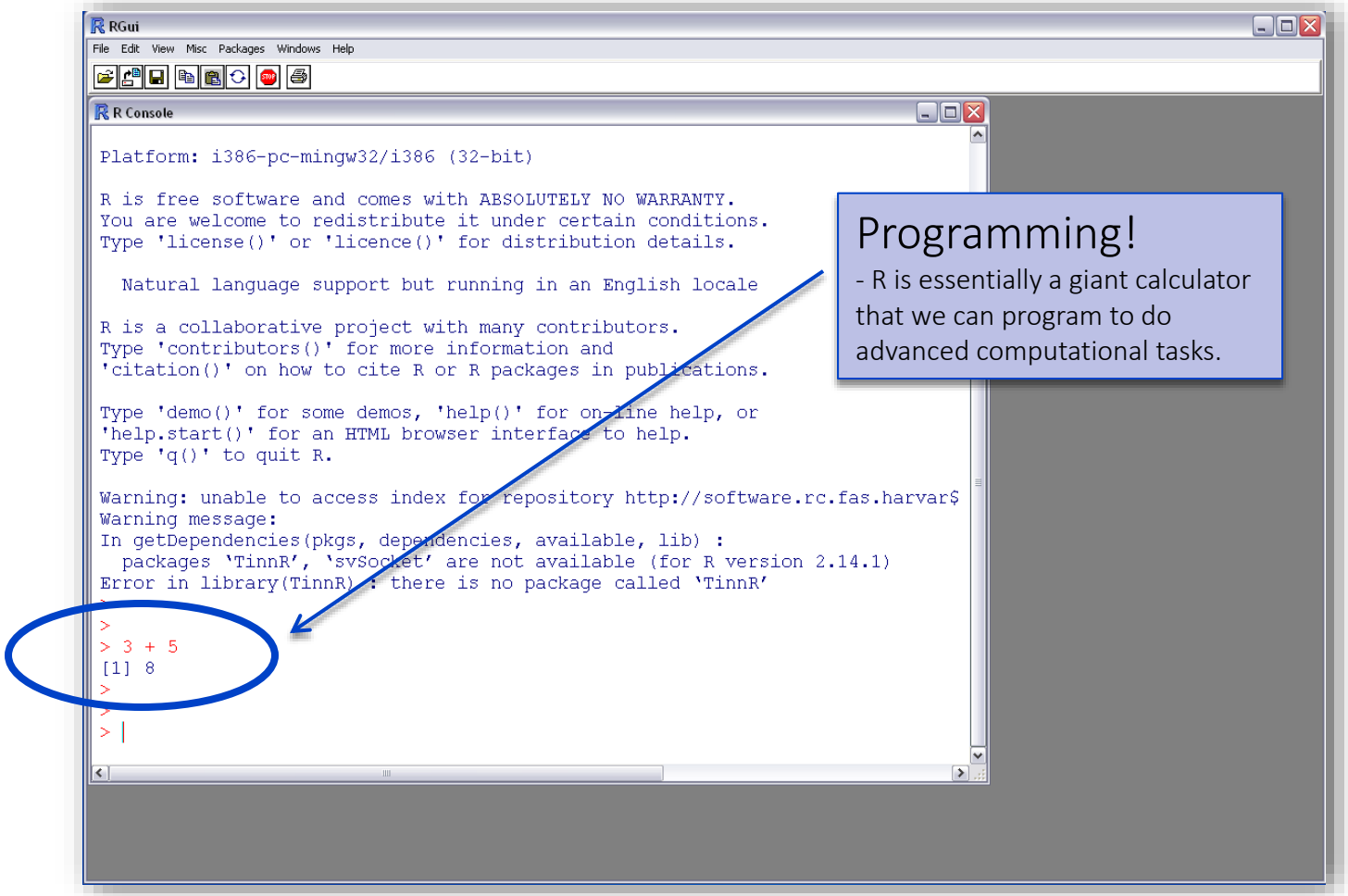
- Objects in R
  - Introduce R and R Studio Software
    - What are the moving parts?
    - How do they work together?
    - What do you need to know to get started?
  - Objects in R
    - Building blocks of programming in R
  - Basic syntax
    - How to create objects in R

What is “Programming”?

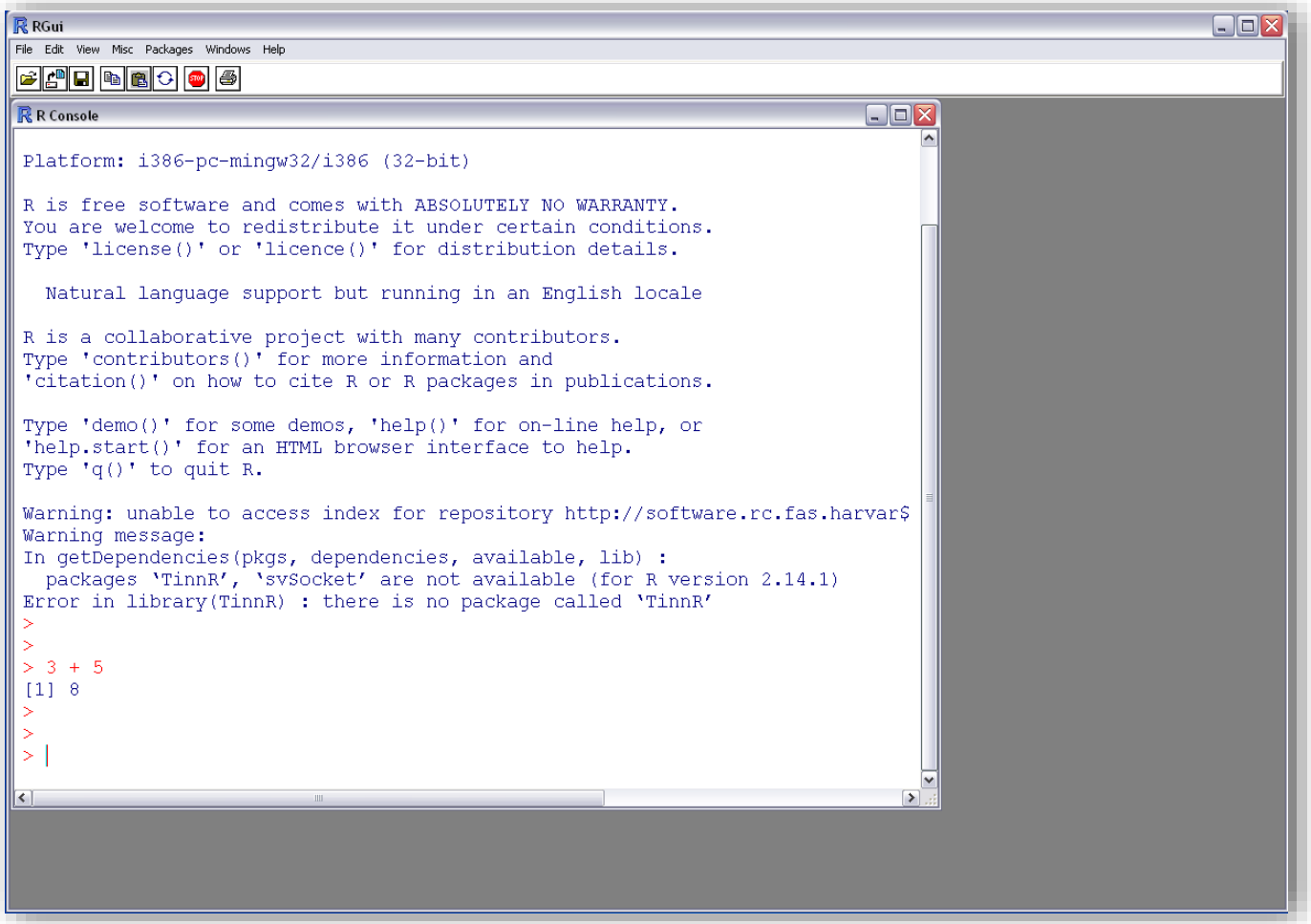
- In the simplest form it is …
  - Writing a line of code
  - Submitting it to R
  - Software processes it
  - Output is returned

- Example| Submit the code `3+5` to R
  - > is the command prompt
  - Type `3+5` and hit Enter/Return to submit
  - R processes the code and returns `[1]8` as the output/answer

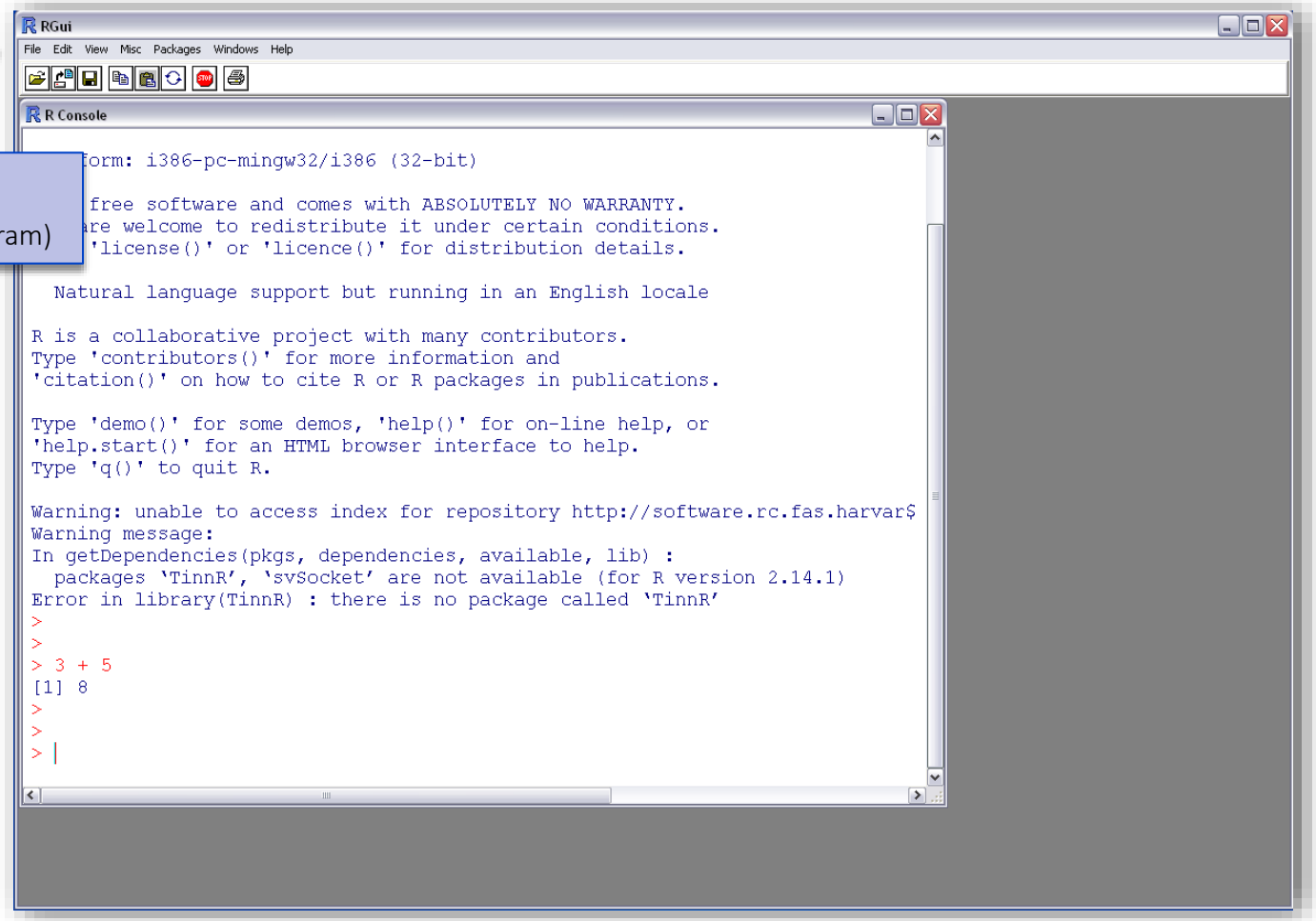
What is “Programming”?



Moving Parts in R

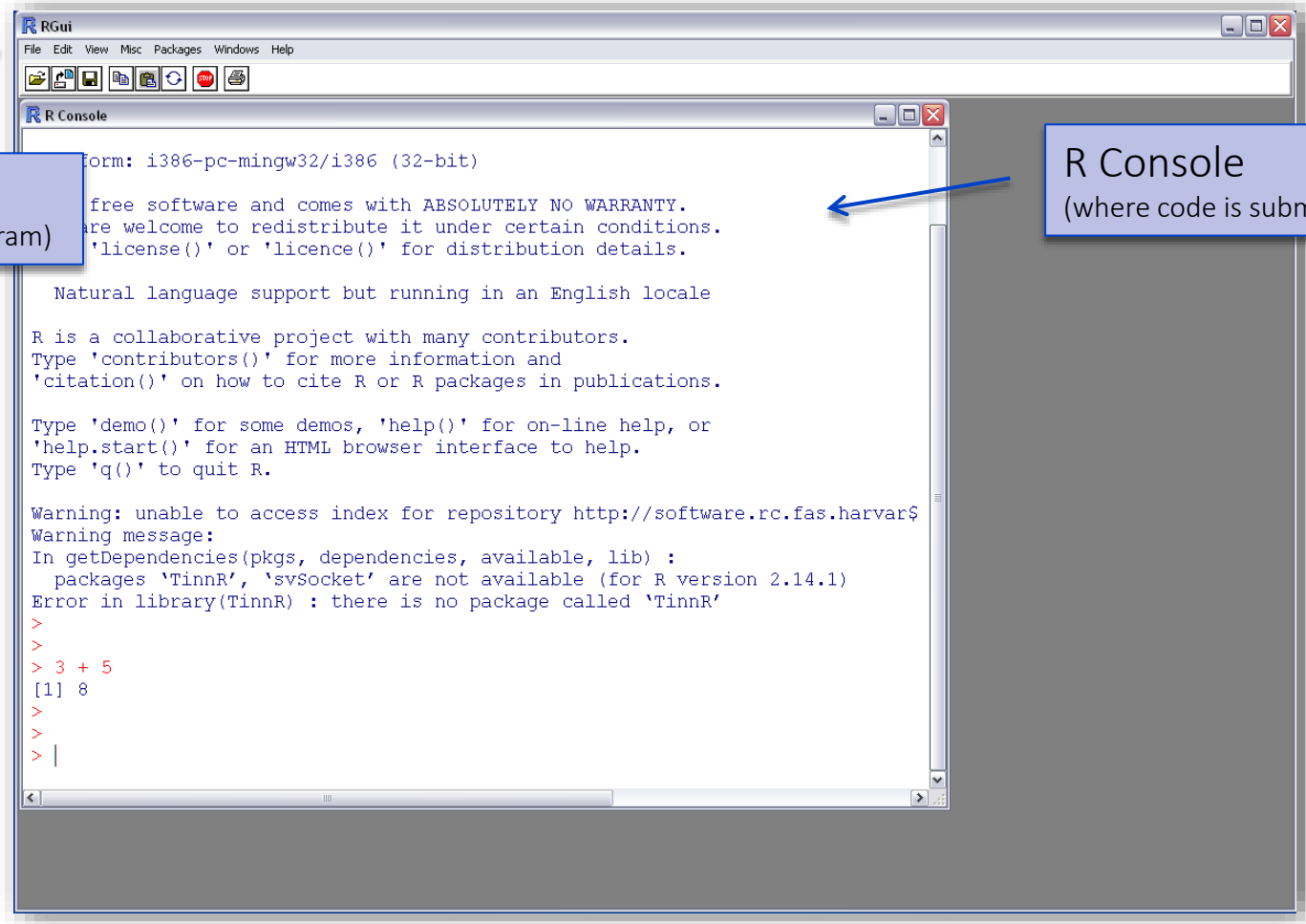


Moving Parts in R



R GUI
(i.e. the software/program)

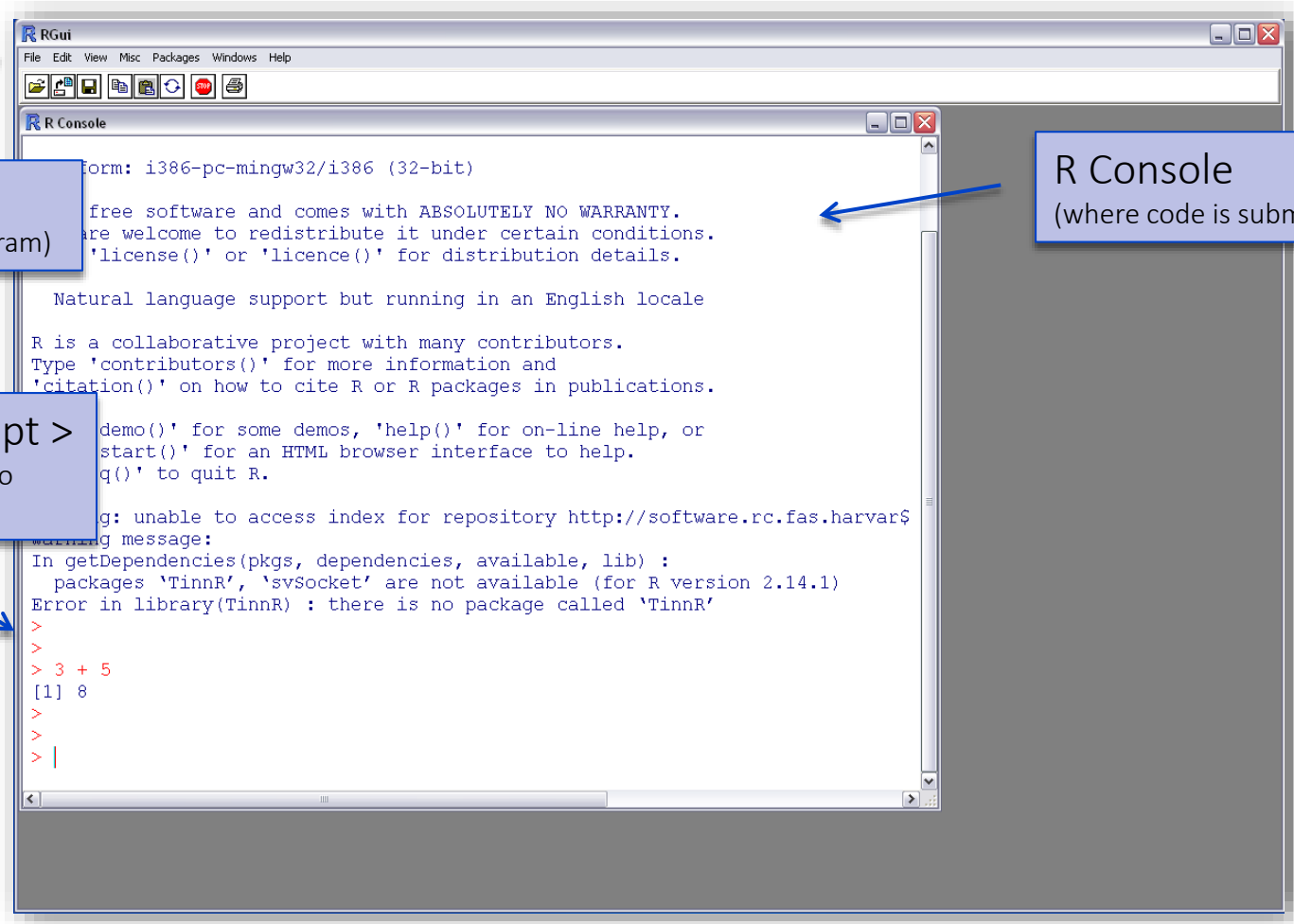
Moving Parts in R



R GUI
(i.e. the software/program)

R Console
(where code is submitted)

Moving Parts in R

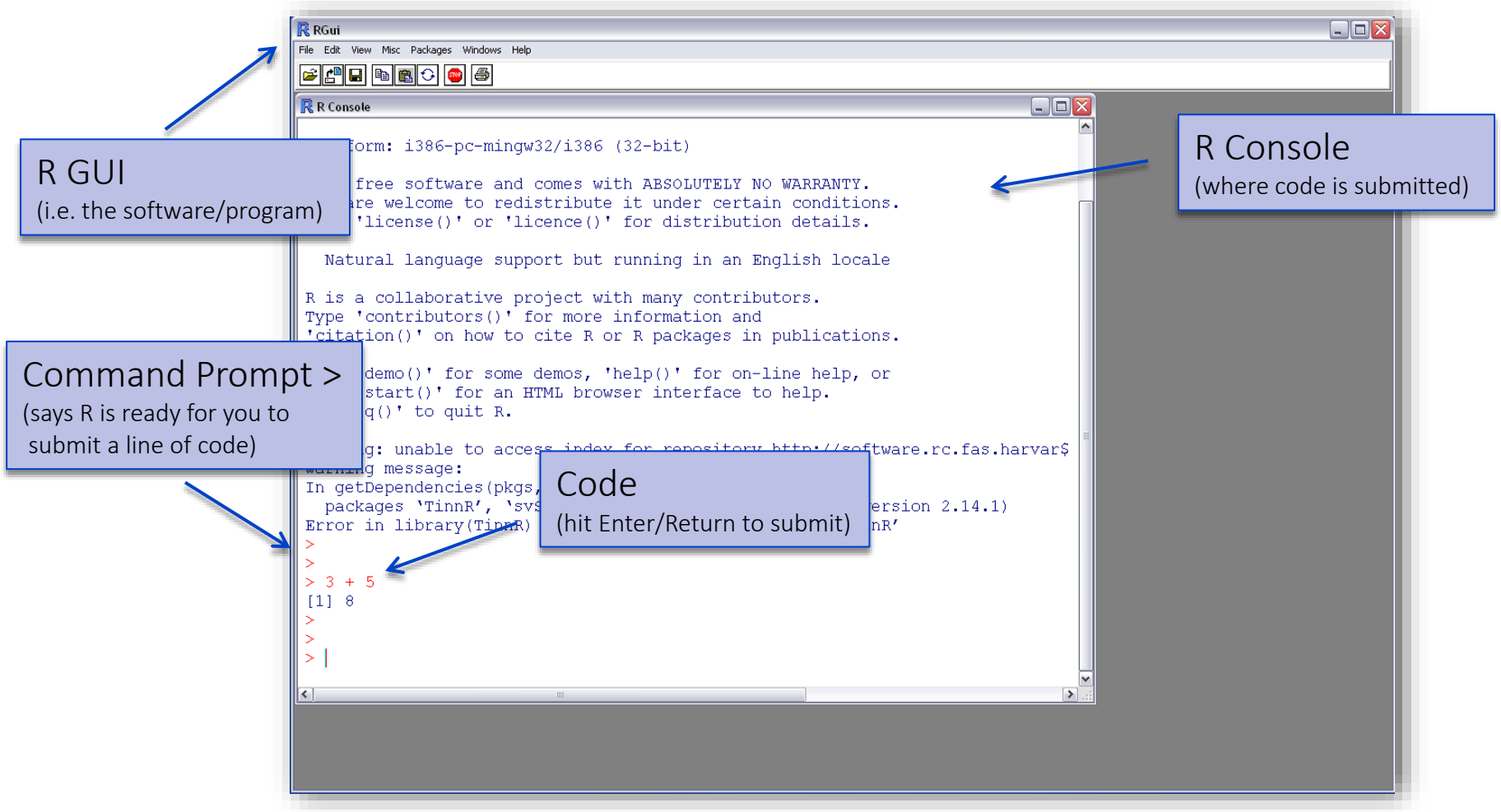


R GUI
(i.e. the software/program)

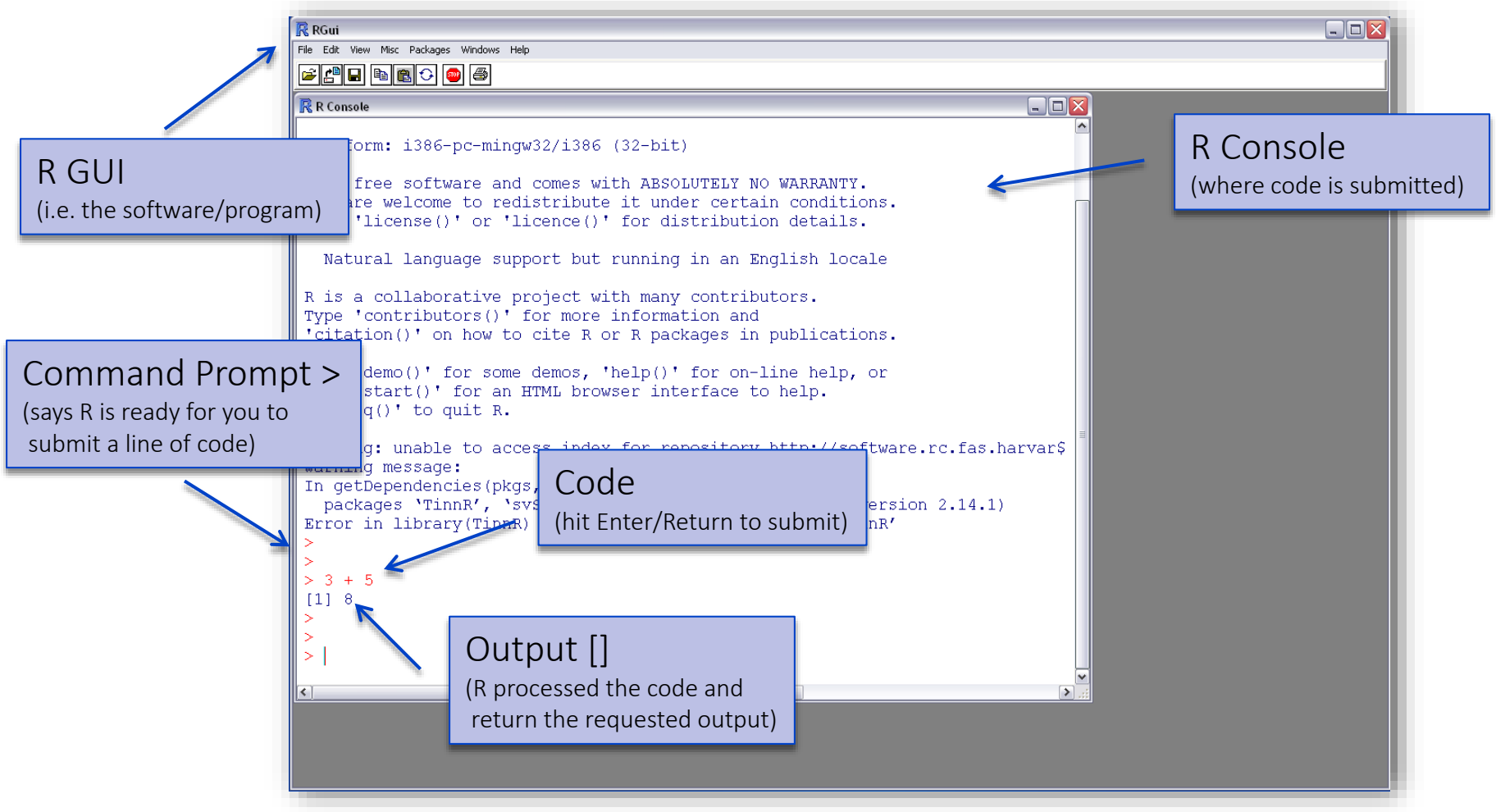
R Console
(where code is submitted)

Command Prompt >
(says R is ready for you to submit a line of code)

Moving Parts in R



Moving Parts in R

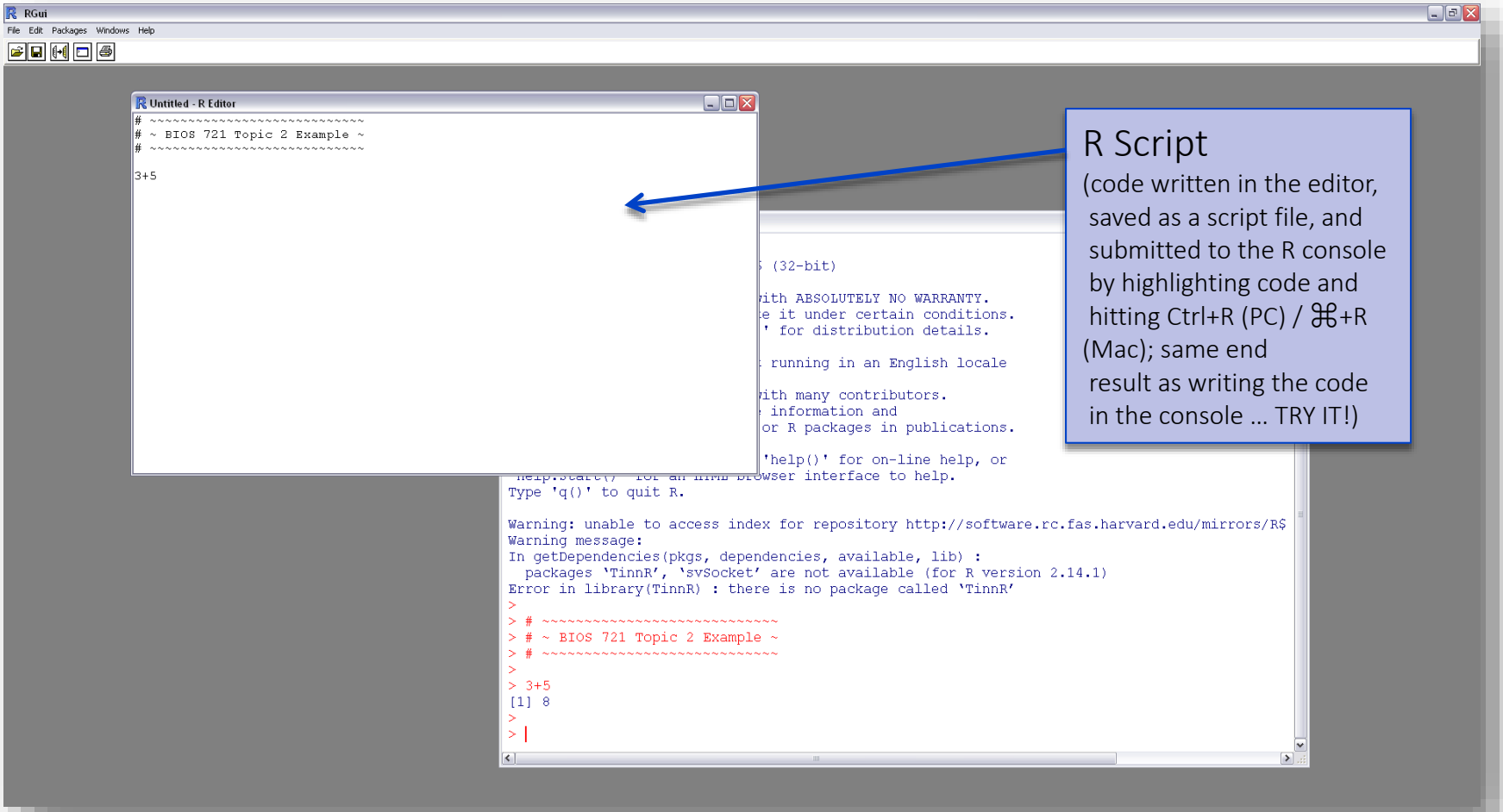


Script Files

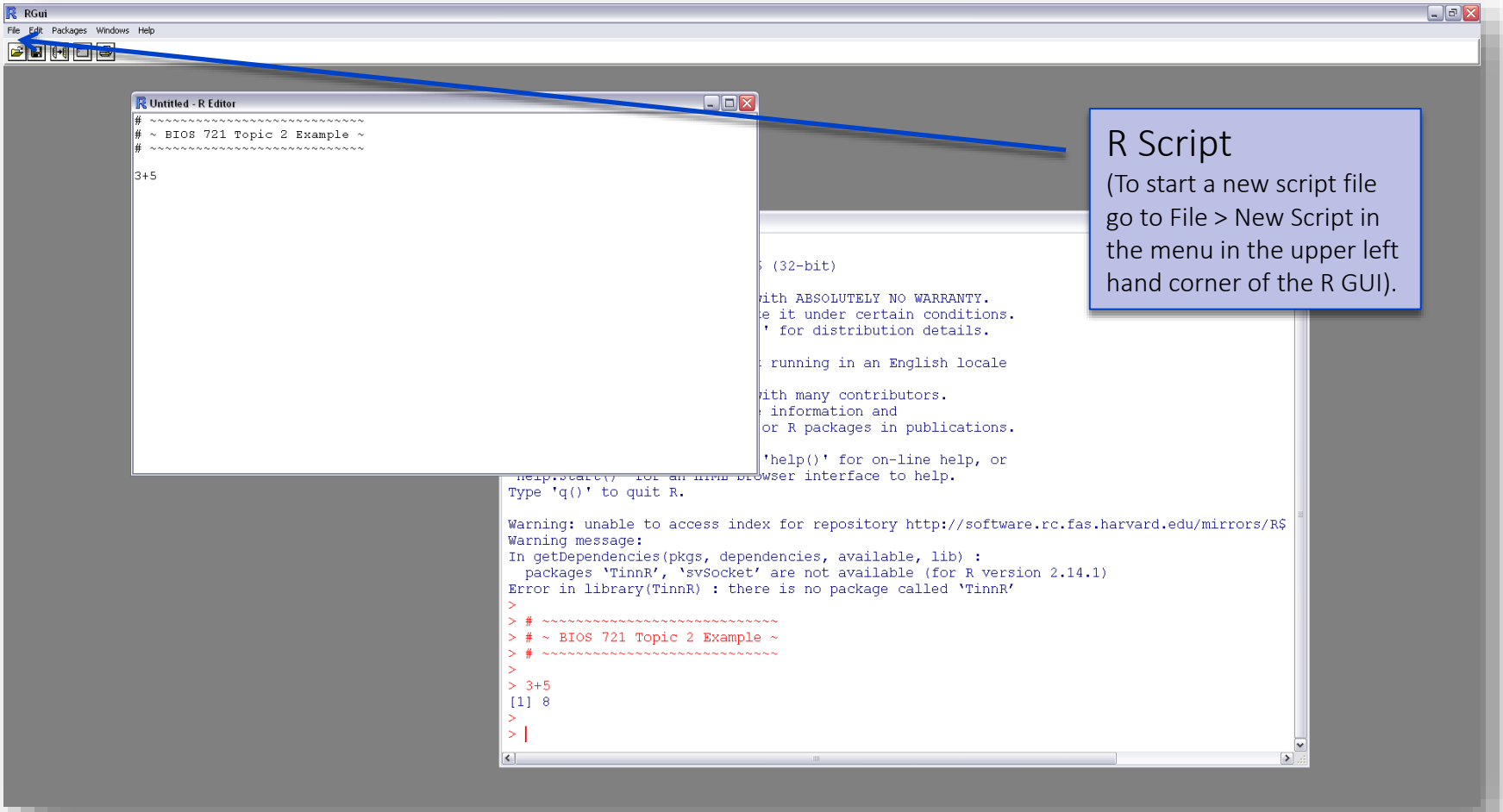
- In the previous example, the “code” was written and submitted in the same place – the R console.
  - In R you do have the option to type code directly into console and submit, but this would become cumbersome if you had A LOT of code to submit and it took A LOT of time to develop (which is typically the case).
  - Instead if code is written and developed using a script file, you can easily save an editable and re-runable version of the code for future use and development.

- For these reasons, using SCRIPT FILES is highly recommended (and somewhat necessary).
  - Scripts contain all of the code you have written in a way that you easily re-submit the code to processor as well make changes to the existing code and add new code to the same file.

Script Files – Use Them!



Script Files – Use Them!



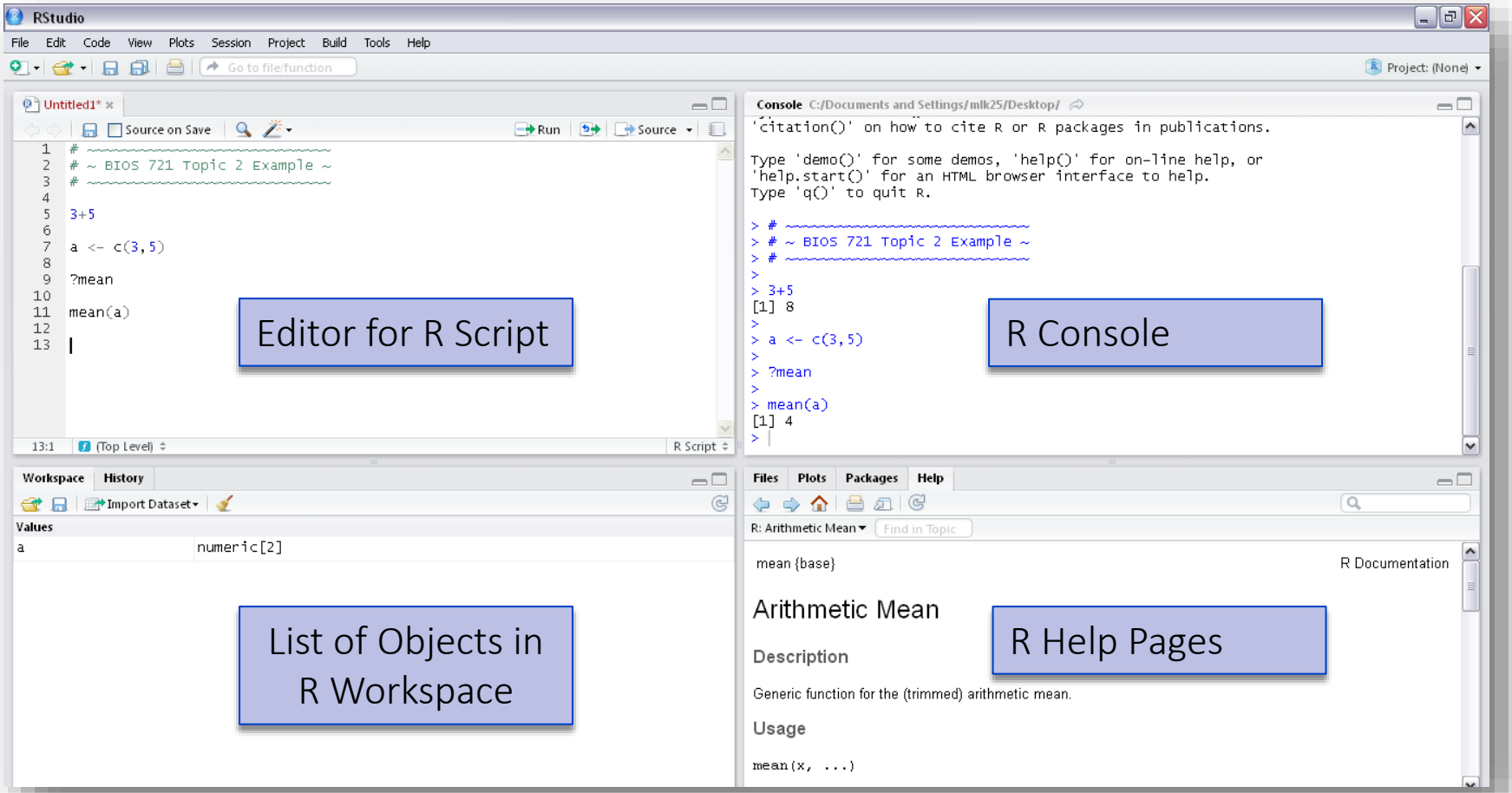
R Script

(To start a new script file go to File > New Script in the menu in the upper left hand corner of the R GUI).

R and R Studio

- Where to code in R and how to submit it?
  - Can do it all within the RGui
    - File > New Script for an editor
    - Highlight code and hit Crtl+R/⌘+R to submit code to R console
  - But I prefer an IDE called R Studio
    - It puts all of the moving parts into a single window (yeah!)
    - Its editor is color coded and has informative highlighting
    - What I will use in this course
      - But you may use whatever you prefer!
  - In all editors, scripts saved as .R files!

Moving Parts in R Studio



Editor for R Script

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 721 Topic 2 Example ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

3+5

a <- c(3,5)

?mean

mean(a)

```

R Console

```
'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.

> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 721 Topic 2 Example ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> 3+5
[1] 8
>
> a <- c(3,5)
>
> ?mean
>
> mean(a)
[1] 4
>
```

List of Objects in R Workspace

Values

a	numeric[2]

R Help Pages

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

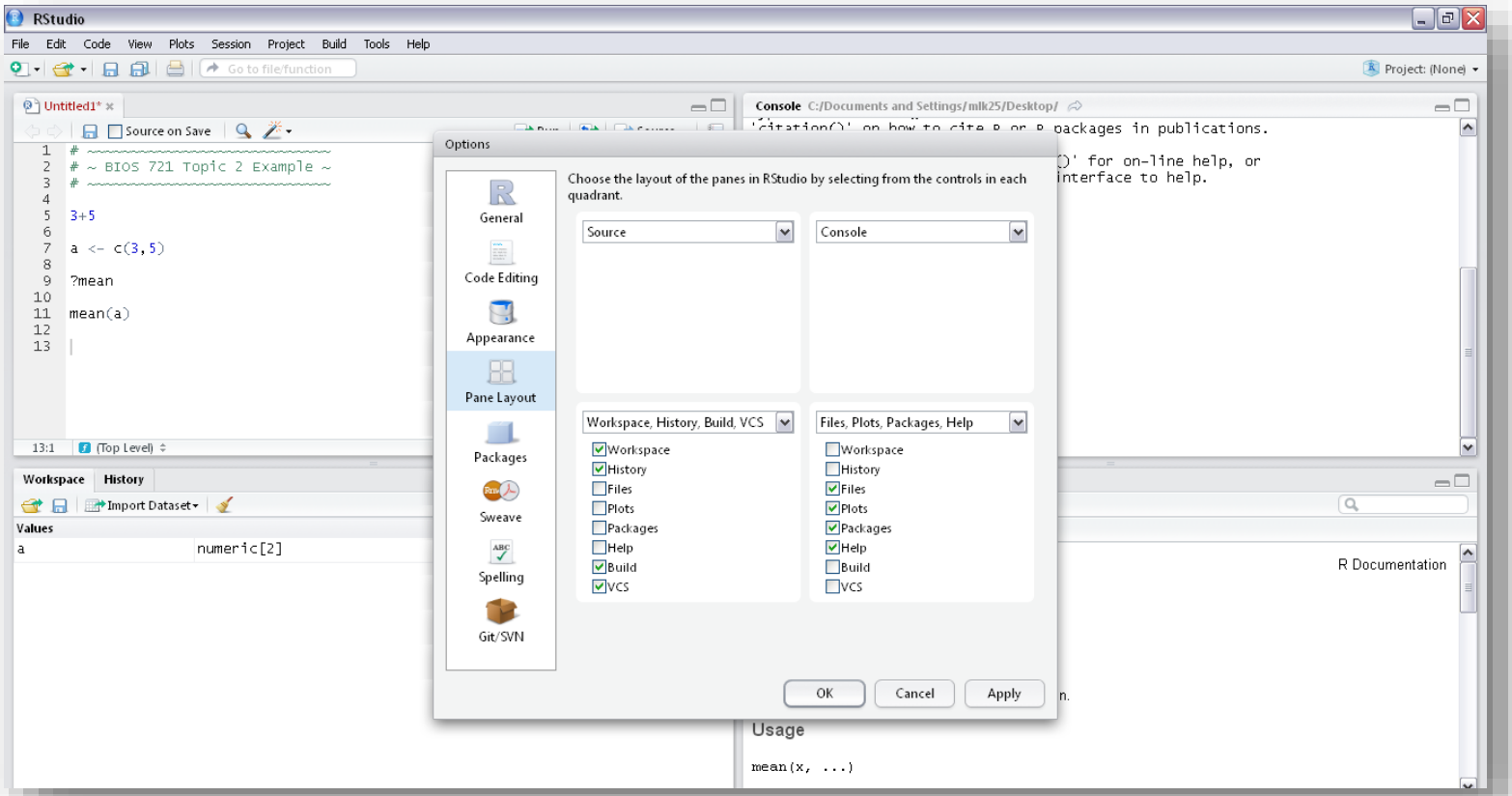
Setting Up R Studio

- Note: You MUST have R installed before using R Studio.

- After installing R Studio, there are a few options you should set to make working a little easier:
  - You may want to re-arrange the window layout (the layout on the previous screen is NOT the default, but is what I prefer).
  - You should set the default working directory (where input will be read from and where output will go by default – I like to use my desktop – you can always change this later)

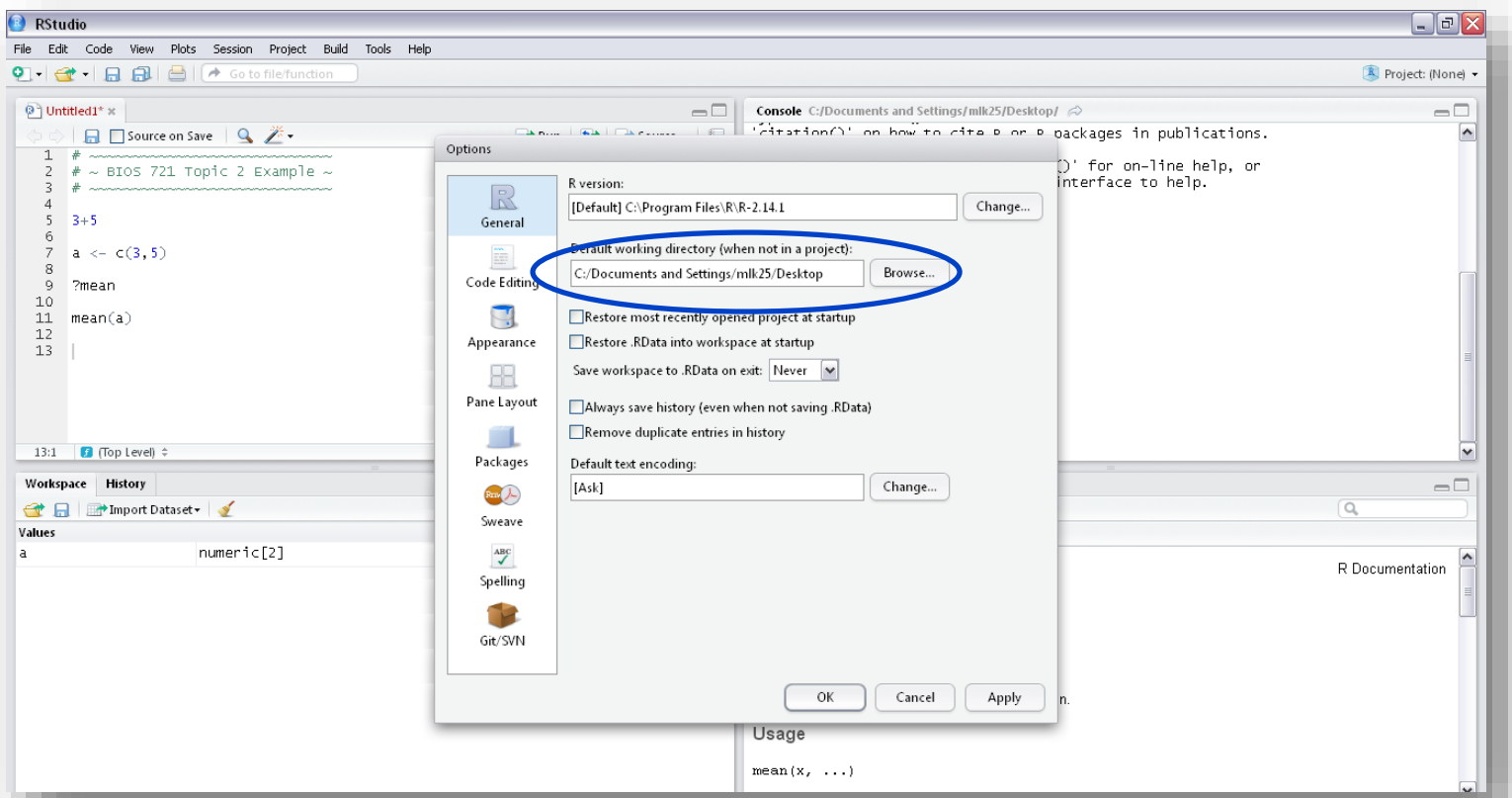
Setting Up R Studio

- Re-arranging the window layout:
  - Tools > Global Options > Pane Layout > Apply/Ok



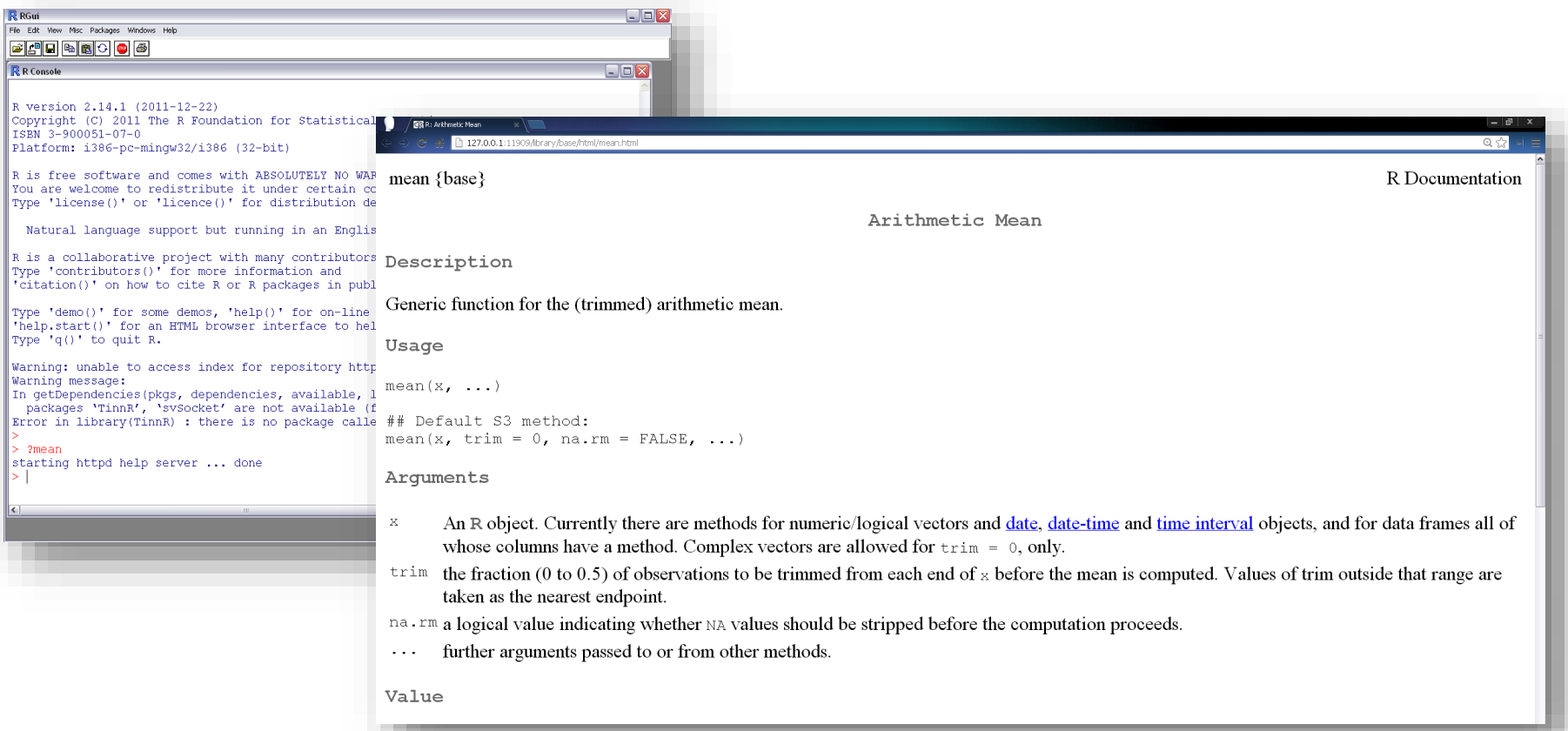
Setting Up R Studio

- Set default working directory:
  - Tools > Options > R General > Enter directory in 2nd box > Apply/Ok
  - I usually use my desktop as my default working directory



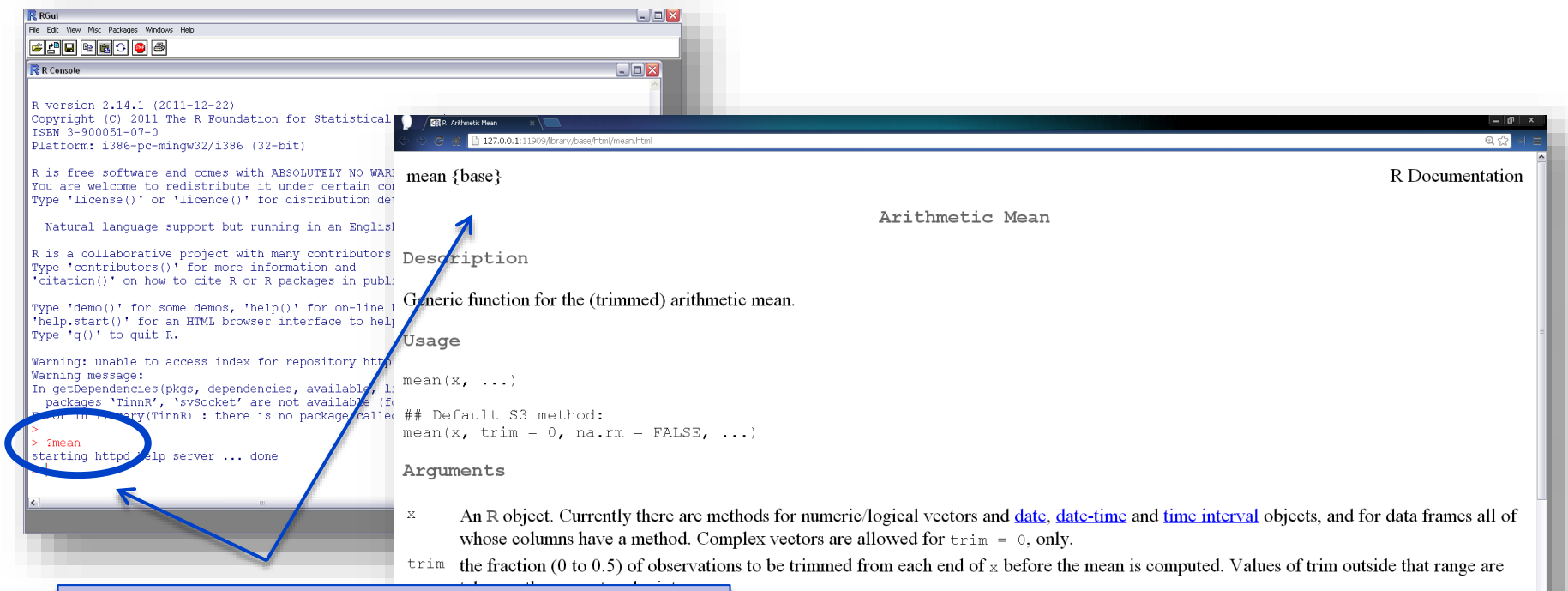
R Help Pages

- By typing `?name_of_function` in the R console, an R help page will be brought up in your internet browser.



R Help Pages

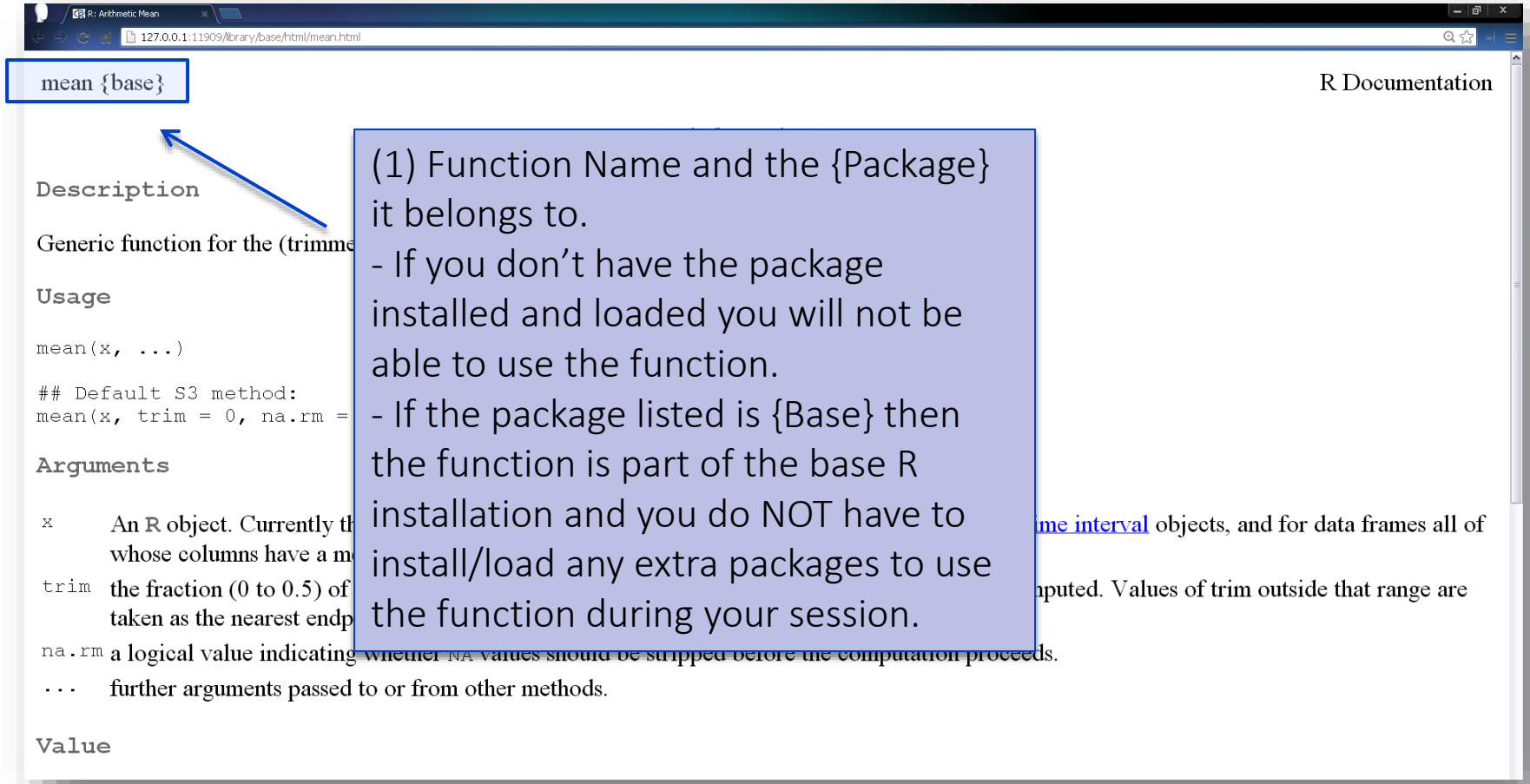
- By typing `?name_of_function` in the R console, an R help page will be brought up in your internet browser. (TRY IT!)



Typing `?mean` into the R console brings up the R help page for the `mean()` function.

R Help Pages

- All R Help Pages have the same basic components:



(1) Function Name and the {Package} it belongs to.
- If you don't have the package installed and loaded you will not be able to use the function.
- If the package listed is {Base} then the function is part of the base R installation and you do NOT have to install/load any extra packages to use the function during your session.

R Help Pages

- All R Help Pages have the same basic components:

R: Arithmetic Mean

127.0.0.1:11909/library/base/html/mean.html

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects, and for data frames all of whose columns have a method. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

(2) Description
- Brief description of what the function does

R Help Pages

- All R Help Pages have the same basic components:

R: Arithmetic Mean

127.0.0.1:11909/library/base/html/mean.html

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors of whose columns have a method. Complex vectors are allowed for `tri`
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end o taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

(3) Usage
- Example call of the function (syntax)
- Lists the required inputs (here, x)
- … indicates that there are other <u>optional</u> inputs that have been set to DEFAULT values.

R Help Pages

- All R Help Pages have the same basic components:

R: Arithmetic Mean

127.0.0.1:11909/library/base/html/mean.html

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

(4) Arguments
- Detailed description of all possible inputs to the function

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects, and for data frames all of whose columns have a method. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

R Help Pages

- All R Help Pages have the same basic components:

R: Arithmetic Mean | Quick-R: Boxplots

127.0.0.1:11909/library/base/html/mean.html

taken as the nearest endpoint.

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

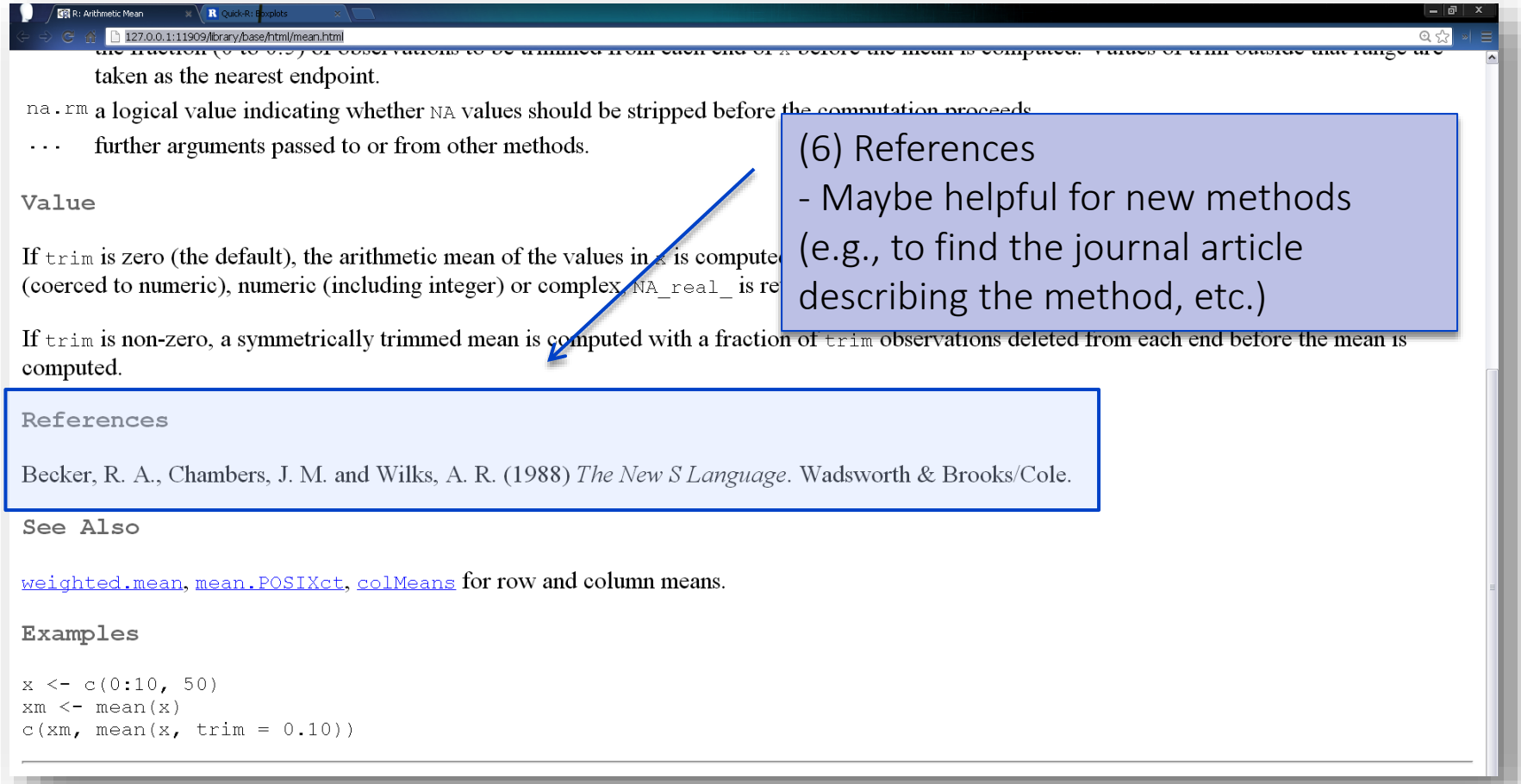
Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

(5) Value
- Detailed description of all possible outputs of the function

R Help Pages

- All R Help Pages have the same basic components:



R Help Pages

- All R Help Pages have the same basic components:

R: Arithmetic Mean | Quick-R: Boxplots

127.0.0.1:11909/library/base/html/mean.html

taken as the nearest endpoint.

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is compute
(coerced to numeric), numeric (including integer) or complex, `NA_real_` is re

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fractio
computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

(7) See Also
- List of other R functions that may be helpful or are used by the current function

R Help Pages

- All R Help Pages have the same basic components:

R: Arithmetic Mean | Quick-R: Boxplots

127.0.0.1:11909/library/base/html/mean.html

taken as the nearest endpoint.

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

(8) Examples
- Run these! Can be useful for learning how the function works.

R Help Pages

- All R Help Pages have the same basic components
- But not all R Help Pages are created equally
  - Some have way better documentation/examples than others
  - Google is your friend – chances are someone else has already done what you are tying to do!
- Another good R resource is the Quick R website
  - http://www.statmethods.net/
  - Maintained by the author of R in Action
  - Help pages are very complete and often have several examples for each function/topic
  - Search “boxplot” and check out what I mean!

R Objects

- R is an object oriented programming language
  - Makes it very flexible and a powerful programming platform
  - Objects can be very complex (e.g. output from a regression analysis) or very simple (e.g. a matrix).

- In their simplest form, you can think of objects as
  - Container + element
  - Example|
    - Container = Matrix
    - Element = Numbers
    - Object = Numeric Matrix

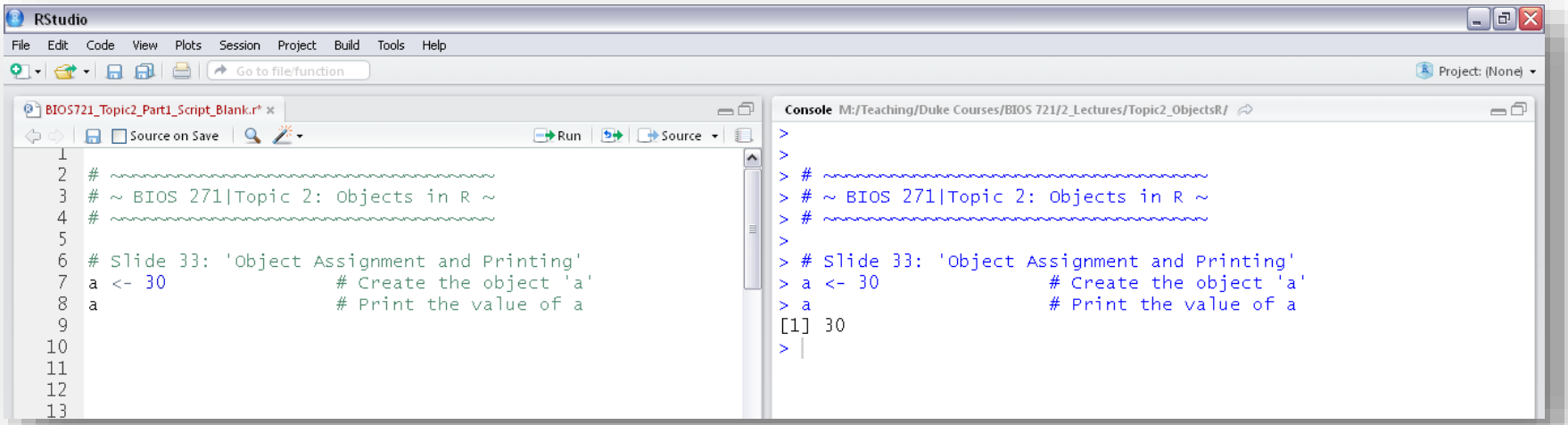
R Objects

- Data Objects / Structures:
  - Scalars
  - Vectors
  - Matrices
  - Arrays
  - Data Frames
  - Lists

- To create an object, use the assignment command:
  - <− or =
- To print (i.e. display the value of) the object, enter the object's name into the console at hit enter.

R Objects

- In the example below,
  - The first line of code creates the object ‘a’ and assigns it to the scalar value 30.
  - The second line of code prints the value of ‘a’.



```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 33: 'Object Assignment and Printing'
a <- 30                 # Create the object 'a'
a                       # Print the value of a
```

```
>
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 33: 'Object Assignment and Printing'
> a <- 30                 # Create the object 'a'
> a                       # Print the value of a
[1] 30
>
```

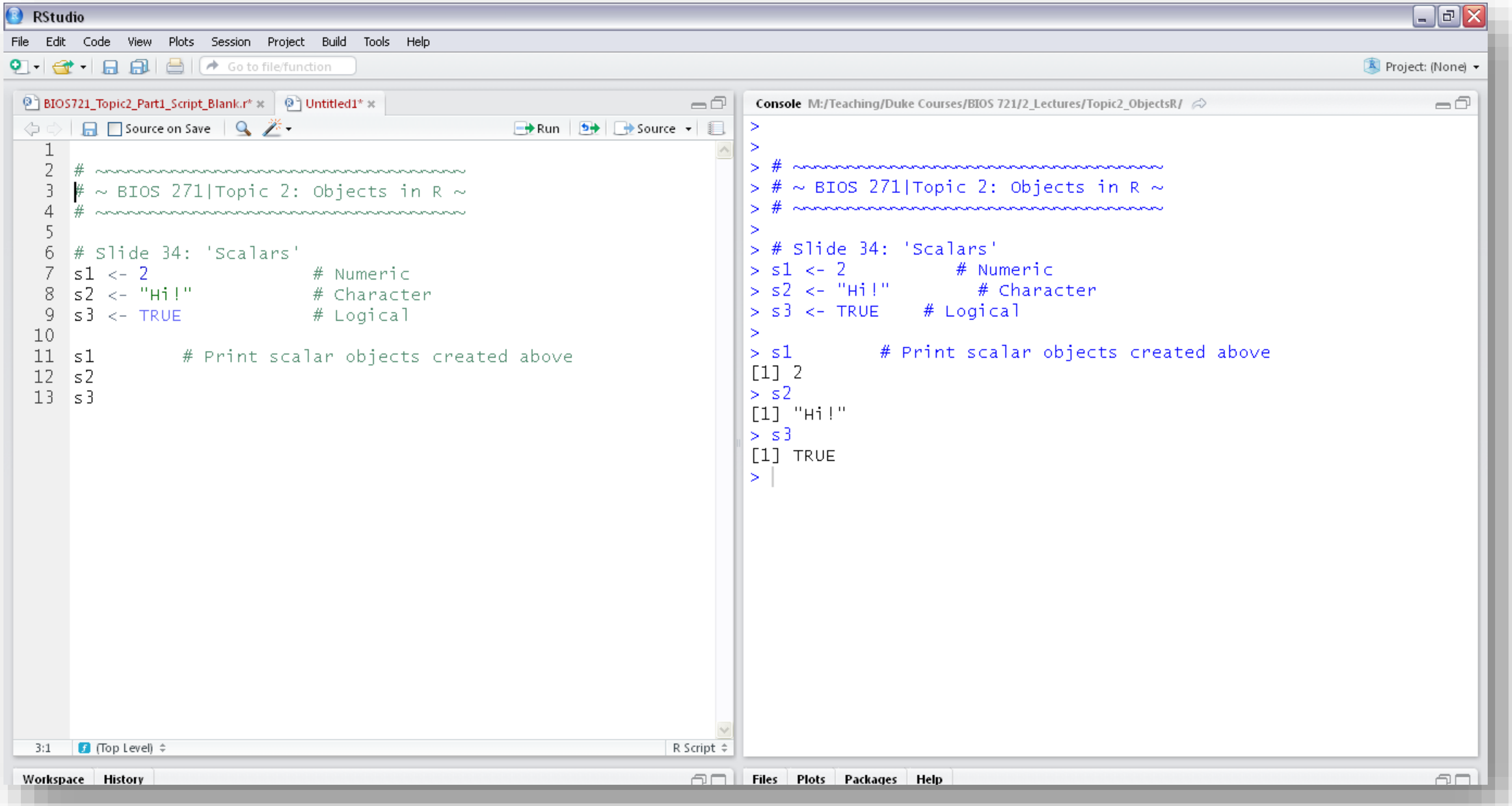
R Objects

- Scalars (a single element)
  - `s1 <- 2` → `[1]2` Numeric
  - `s2 <- "Hi!"` → `[1]"Hi!"` Character
  - `s3 <- TRUE` → `[1]TRUE` Logical

Mode
a.k.a
Type of
Data

- Special Cases
  - `NA` = missing value
  - `NaN` = Not a number
  - `-Inf/Inf` = Infinity (or at least close to it)

R Objects



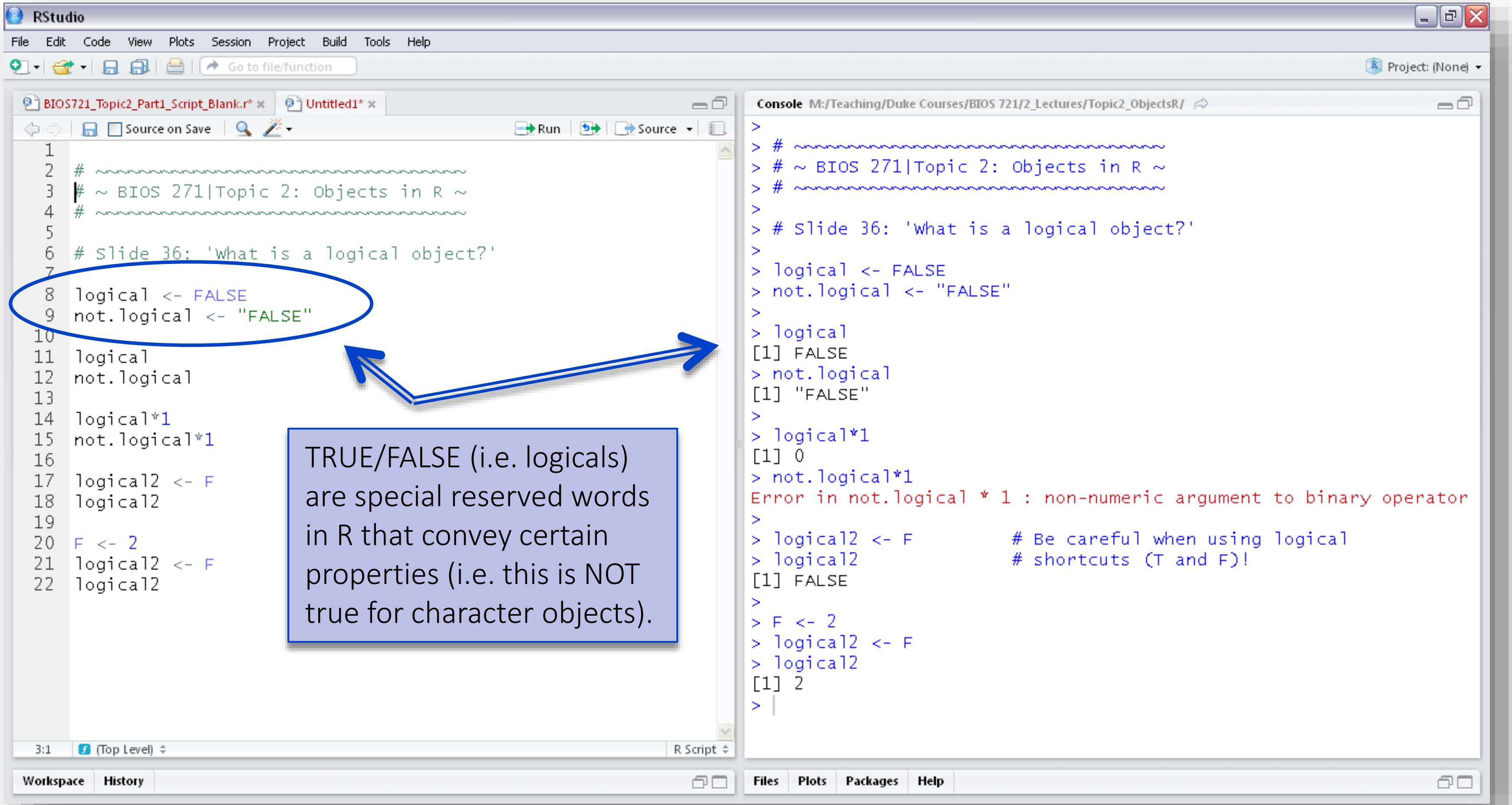
```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 34: 'Scalars'
s1 <- 2                 # Numeric
s2 <- "Hi!"             # Character
s3 <- TRUE              # Logical

s1          # Print scalar objects created above
s2
s3
```

```
>
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 34: 'Scalars'
> s1 <- 2          # Numeric
> s2 <- "Hi!"        # Character
> s3 <- TRUE     # Logical
>
> s1         # Print scalar objects created above
[1] 2
> s2
[1] "Hi!"
> s3
[1] TRUE
>
```

R Objects

- A Note on Logical Objects:
  - `TRUE` and `FALSE` are reserved words (i.e. users cannot assign values to these words)
  - Although the values are "character" visually, they can be manipulated numerically (i.e. R easily allows `TRUE` and `FALSE` to be converted to 1 and 0, respectively).
  - `T` and `F` are shortcuts that can be used in place of `TRUE` and `FALSE`, respectively.
    - But Be Careful! `T` and `F` are not reserved, so users can over write these values, which can lead to errors (see next slide for example).

R Objects

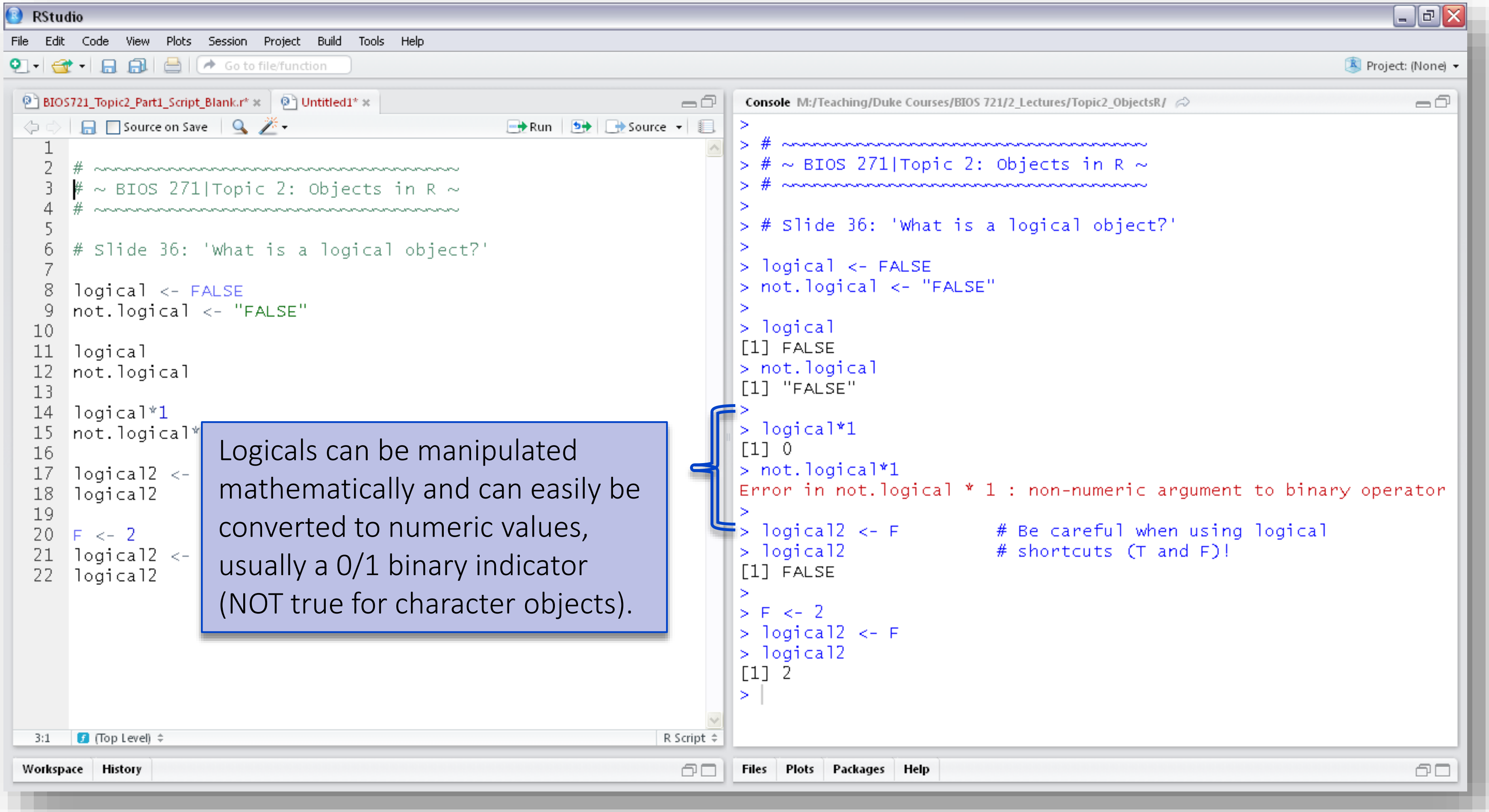


```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 36: 'What is a logical object?'

logical <- FALSE
not.logical <- "FALSE"

logical
not.logical

logical*1
not.logical*1

logical2 <- F
logical2

F <- 2
logical2 <- F
logical2
```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 36: 'What is a logical object?'
>
> logical <- FALSE
> not.logical <- "FALSE"
>
> logical
[1] FALSE
> not.logical
[1] "FALSE"
>
> logical*1
[1] 0
> not.logical*1
Error in not.logical * 1 : non-numeric argument to binary operator
>
> logical2 <- F         # Be careful when using logical
> logical2              # shortcuts (T and F)!
[1] FALSE
>
> F <- 2
> logical2 <- F
> logical2
[1] 2
>
```

TRUE/FALSE (i.e. logicals) are special reserved words in R that convey certain properties (i.e. this is NOT true for character objects).

R Objects

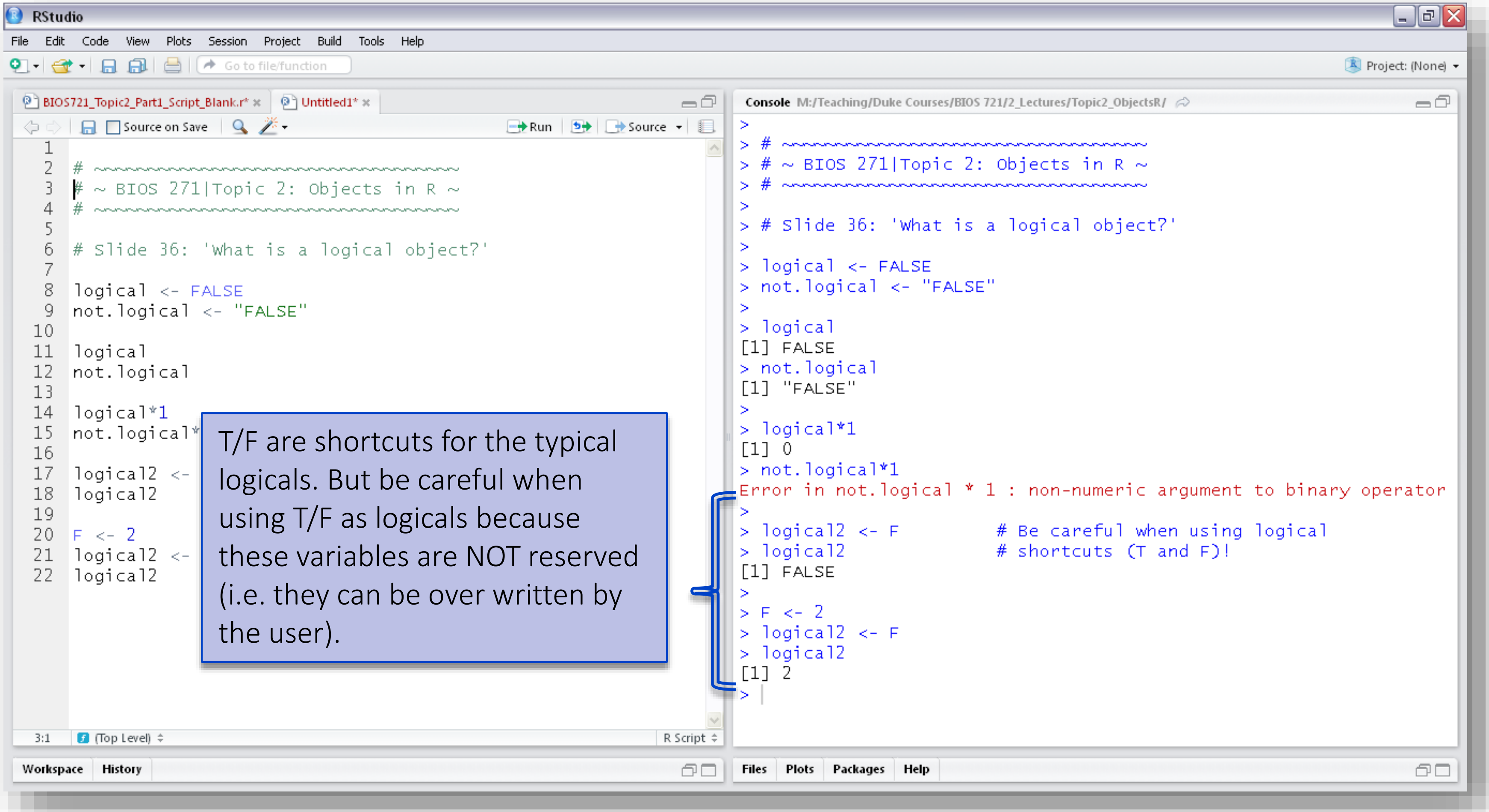


```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 36: 'What is a logical object?'

logical <- FALSE
not.logical <- "FALSE"

logical
not.logical

logical*1
not.logical*

logical2 <-
logical2

F <- 2
logical2 <-
logical2
```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 36: 'What is a logical object?'
>
> logical <- FALSE
> not.logical <- "FALSE"
>
> logical
[1] FALSE
> not.logical
[1] "FALSE"
>
> logical*1
[1] 0
> not.logical*1
Error in not.logical * 1 : non-numeric argument to binary operator
>
> logical2 <- F          # Be careful when using logical
> logical2               # shortcuts (T and F)!
[1] FALSE
>
> F <- 2
> logical2 <- F
> logical2
[1] 2
>
```

Logicals can be manipulated mathematically and can easily be converted to numeric values, usually a 0/1 binary indicator (NOT true for character objects).

R Objects



```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 36: 'What is a logical object?'

logical <- FALSE
not.logical <- "FALSE"

logical
not.logical

logical*1
not.logical*

logical2 <-
logical2

F <- 2
logical2 <-
logical2
```

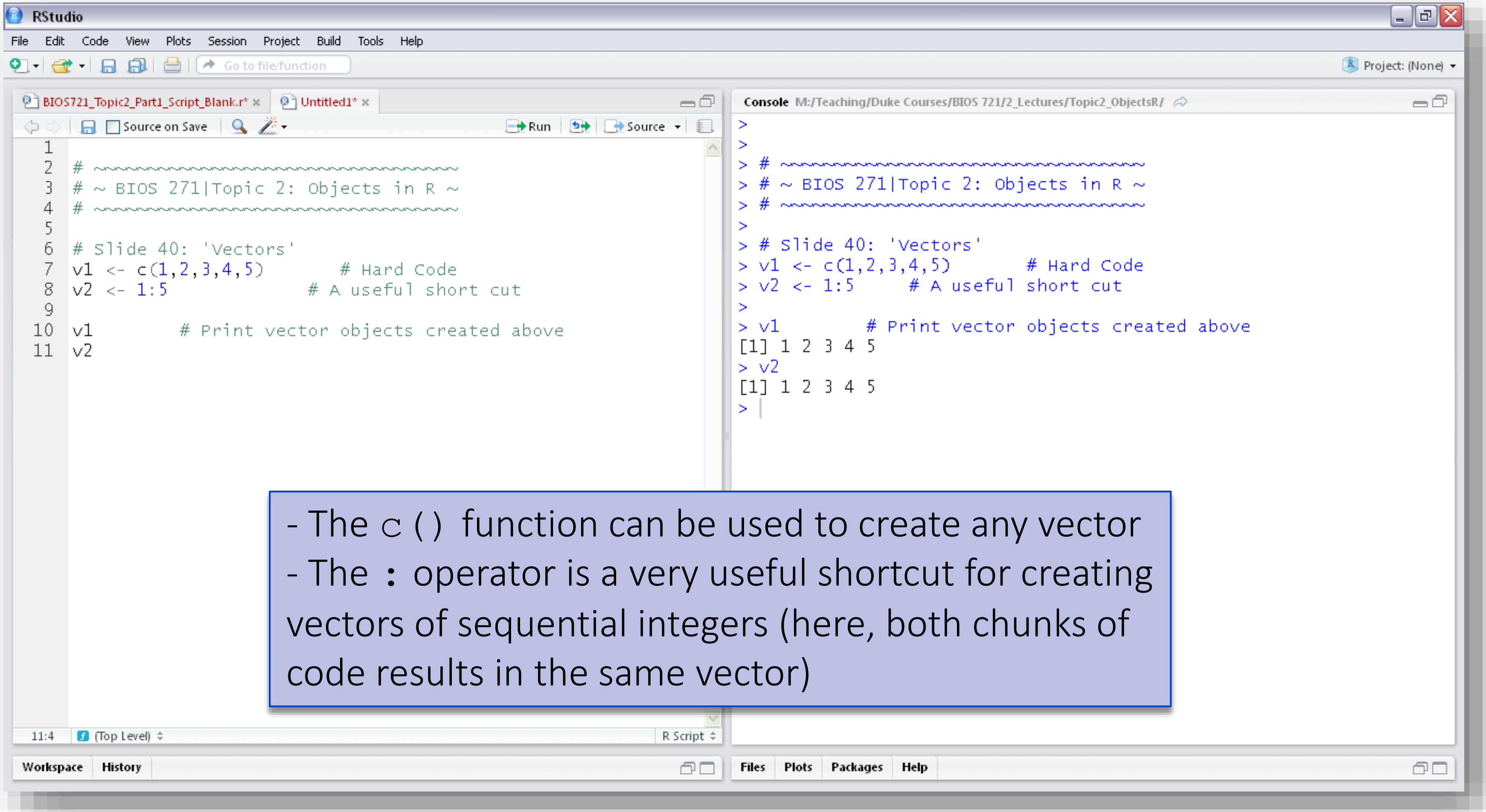
```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 36: 'What is a logical object?'
>
> logical <- FALSE
> not.logical <- "FALSE"
>
> logical
[1] FALSE
> not.logical
[1] "FALSE"
>
> logical*1
[1] 0
> not.logical*1
Error in not.logical * 1 : non-numeric argument to binary operator
>
> logical2 <- F          # Be careful when using logical
> logical2               # shortcuts (T and F)!
[1] FALSE
>
> F <- 2
> logical2 <- F
> logical2
[1] 2
>
```

T/F are shortcuts for the typical logicals. But be careful when using T/F as logicals because these variables are NOT reserved (i.e. they can be over written by the user).

R Objects

- Vectors (stacks of scalars)
  - Create using the R 'combine' function: `c(,…)`
  - Hard Code
    - `v1 <- c(1,2,3,4,5)` → `[1] 1 2 3 4 5`
  - But there are lots of shortcuts … (more in the PPs)
    - `v2 <- 1:5` → `[1] 1 2 3 4 5`

R Objects



```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 40: 'Vectors'
v1 <- c(1,2,3,4,5)        # Hard Code
v2 <- 1:5              # A useful short cut

v1        # Print vector objects created above
v2
```

```
>
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 40: 'Vectors'
> v1 <- c(1,2,3,4,5)        # Hard Code
> v2 <- 1:5     # A useful short cut
>
> v1        # Print vector objects created above
[1] 1 2 3 4 5
> v2
[1] 1 2 3 4 5
>
```

- The `c()` function can be used to create any vector
- The **:** operator is a very useful shortcut for creating vectors of sequential integers (here, both chunks of code results in the same vector)

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
        byrow=FALSE,dimnames=NULL)
```

OR

```
matrix(data,nrow,ncol,...)
```

  - Examples
    - `M1 <- matrix(1:6,3,2)`
    - `M2 <- matrix(1:6,3,2,byrow=TRUE)`
    - `M3 <- matrix(1:6,3,2, byrow=TRUE,`
      `dimnames=list(c("R1", "R2","R3"),`
      `c("C1","C2")))`

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
        byrow=FALSE,dimnames=NULL)
```

OR

```
matrix(data,nrow,ncol,...)
```

Recall:
- Any arguments NOT assigned a value are required input for the function to excute (here data, nrow, and ncol)
- Any arguments assigned to a value are optional; if not given a value during the function call, they will be set to the default value (here byrow and dimnames)
- To learn about a function, check out the R help page

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
       byrow=FALSE,dimnames=NULL)
```

OR

data = any VECTOR of values (numeric, character, or logical)

  - Examples
    - `M1 <- matrix(1:6,3,2)`
    - `M2 <- matrix(1:6,3,2,byrow=TRUE)`
    - `M3 <- matrix(1:6,3,2, byrow=TRUE,`
      `dimnames=list(c("R1", "R2","R3"),`
      `c("C1","C2")))`

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
       byrow=FALSE,dimnames=NULL)
```

OR

nrow = number of rows in the matrix

  - Examples
    - `M1 <- matrix(1:6,3,2)`
    - `M2 <- matrix(1:6,3,2,byrow=TRUE)`
    - `M3 <- matrix(1:6,3,2, byrow=TRUE,`
      `dimnames=list(c("R1", "R2","R3"),`
      `c("C1","C2")))`

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
       byrow=FALSE,dimnames=NULL)
```

OR

ncol = number of columns in the matrix

  - Examples
    - `M1 <- matrix(1:6,3,2)`
    - `M2 <- matrix(1:6,3,2,byrow=TRUE)`
    - `M3 <- matrix(1:6,3,2, byrow=TRUE,`
      `dimnames=list(c("R1", "R2","R3"),`
      `c("C1","C2")))`

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
       byrow=FALSE,dimnames=NULL)
```

OR

byrow = logical indicator for whether the vector should fill the matrix down columns (byrow=FALSE) or across rows (byrow=TRUE)

  - Examples
    - `M1 <- matrix(1:6,3,2)`
    - `M2 <- matrix(1:6,3,2,byrow=TRUE)`
    - `M3 <- matrix(1:6,3,2, byrow=TRUE,`
      `dimnames=list(c("R1", "R2","R3"),`
      `c("C1","C2")))`

R Objects

- Matrices (stacks of vectors)
  - Create using the R function 'matrix':

```
matrix(data,nrow,ncol,
        byrow=FALSE,dimnames=NULL)
```

OR

dimnames = a list of two vectors containing the row and column names; row names should be listed first

  - Examples
    - `M1 <- matrix(1:6,3,2)`
    - `M2 <- matrix(1:6,3,2,byrow=TRUE)`
    - `M3 <- matrix(1:6,3,2, byrow=TRUE,`
      `dimnames=list(c("R1", "R2","R3"),`
      `c("C1","C2")))`

R Objects

RStudio

File Edit Code View Plots Session Project Build Tools Help

BIOS721_Topic2_Part1_Script_Blank.r* | Untitled1*

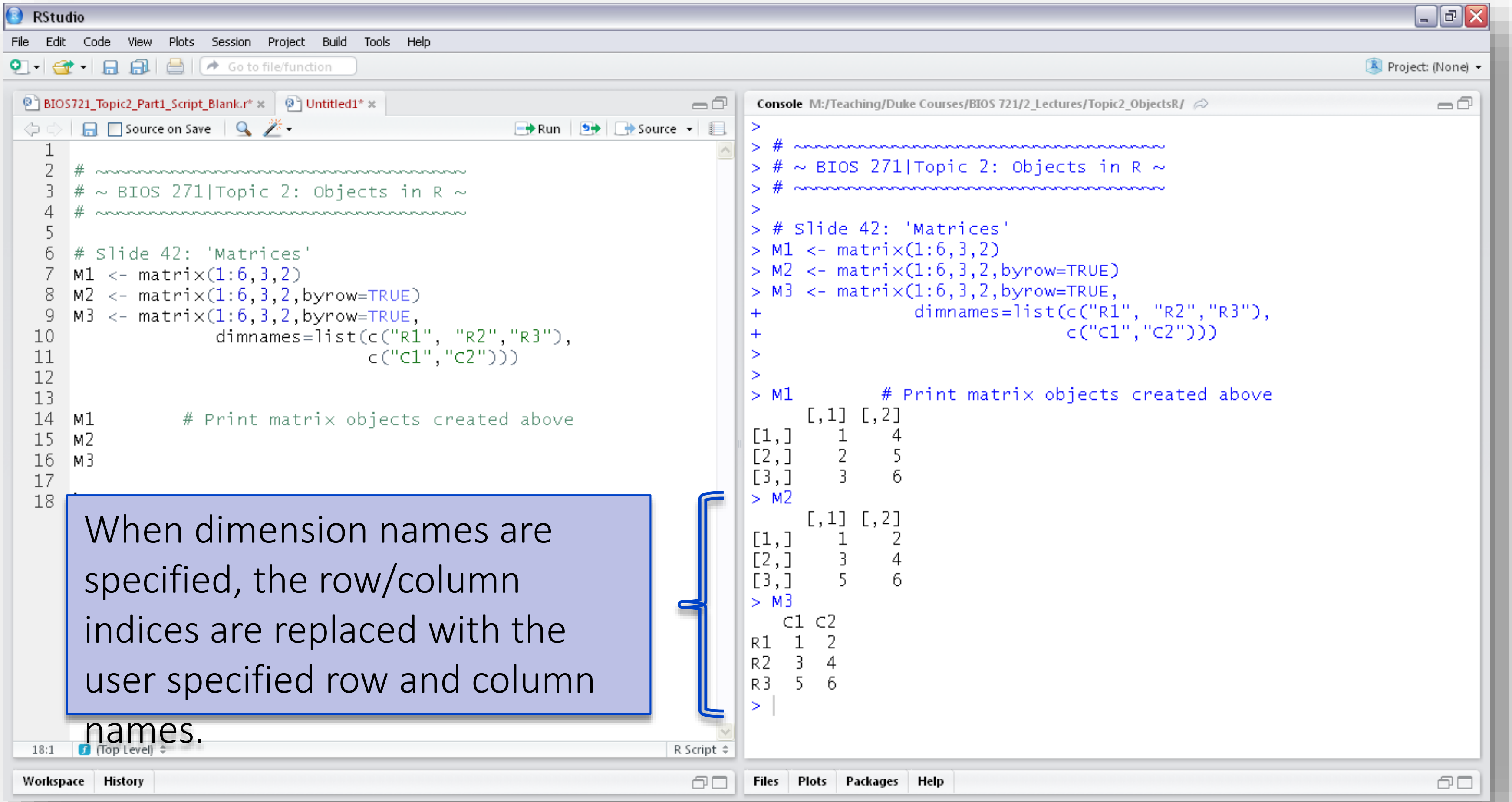
```r
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 42: 'Matrices'
M1 <- matrix(1:6,3,2)
M2 <- matrix(1:6,3,2,byrow=TRUE)
M3 <- matrix(1:6,3,2,byrow=TRUE,
             dimnames=list(c("R1", "R2","R3"),
                           c("C1","C2")))


M1        # Print matrix objects created above
M2
M3
```

Console M:/Teaching/Duke Courses/BIOS 721/2_Lectures/Topic2_ObjectsR/

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 42: 'Matrices'
> M1 <- matrix(1:6,3,2)
> M2 <- matrix(1:6,3,2,byrow=TRUE)
> M3 <- matrix(1:6,3,2,byrow=TRUE,
+              dimnames=list(c("R1", "R2","R3"),
+                            c("C1","C2")))
>
>
> M1        # Print matrix
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> M2
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> M3
   C1 C2
R1  1  2
R2  3  4
R3  5  6
>
```

Compare the results when byrow=FALSE vs. byrow=TRUE

R Objects



```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 42: 'Matrices'
M1 <- matrix(1:6,3,2)
M2 <- matrix(1:6,3,2,byrow=TRUE)
M3 <- matrix(1:6,3,2,byrow=TRUE,
             dimnames=list(c("R1", "R2","R3"),
                           c("C1","C2")))


M1        # Print matrix objects created above
M2
M3
```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 42: 'Matrices'
> M1 <- matrix(1:6,3,2)
> M2 <- matrix(1:6,3,2,byrow=TRUE)
> M3 <- matrix(1:6,3,2,byrow=TRUE,
+              dimnames=list(c("R1", "R2","R3"),
+                            c("C1","C2")))
>
>
> M1        # Print matrix objects created above
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> M2
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> M3
   C1 C2
R1  1  2
R2  3  4
R3  5  6
>
```

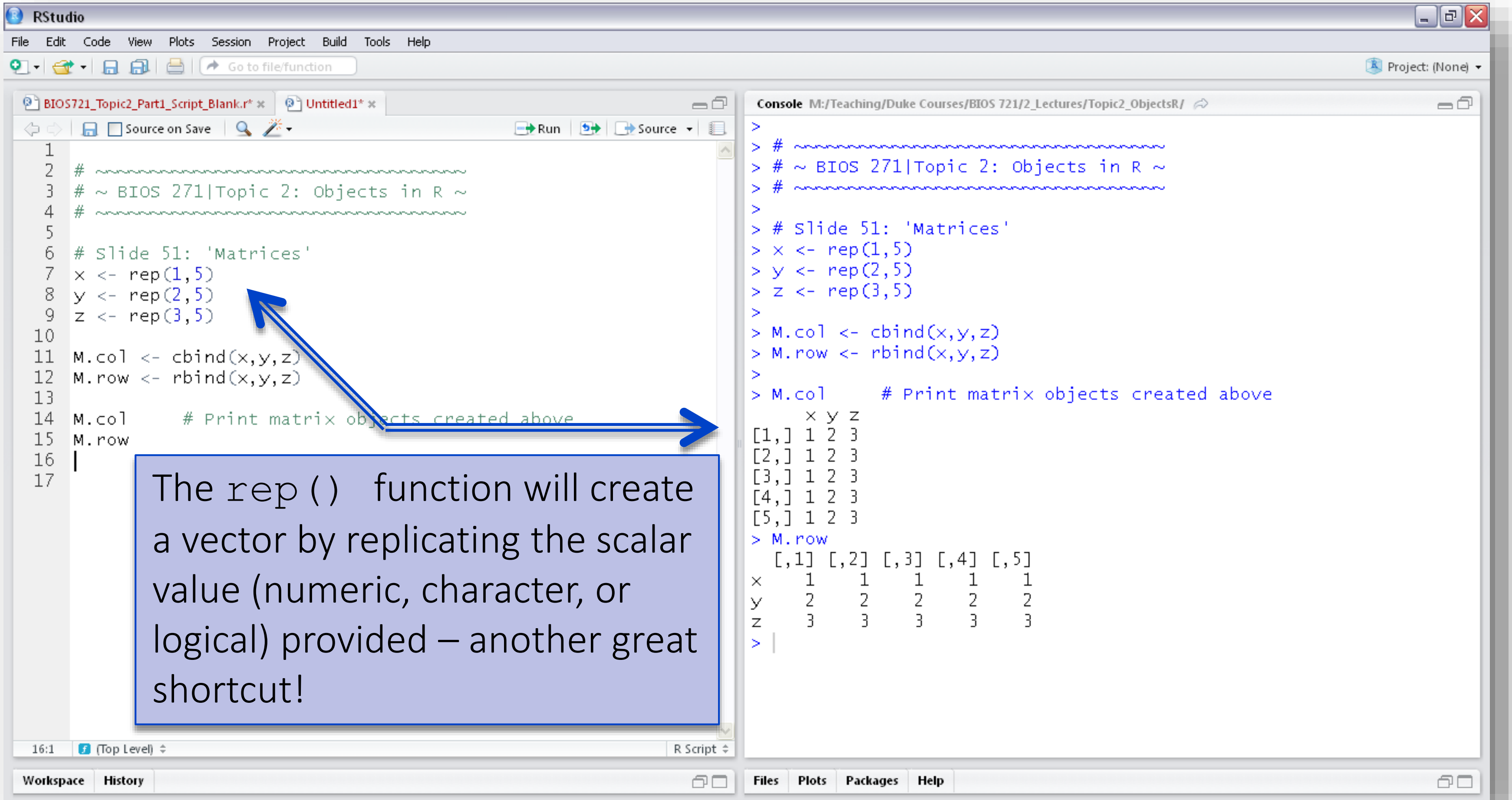
When dimension names are specified, the row/column indices are replaced with the user specified row and column names.

R Objects

- Matrices (stacks of vectors)
  - Can also create matrices by combining vectors by rows or by columns (can be very useful in simulations)
  - Examples
    - `x <- rep(1,5)`
    - `y <- rep(2,5)`
    - `z <- rep(3,5)`

    - `M.col <- cbind(x,y,z)`
    - `M.row <- rbind(x,y,z)`

R Objects

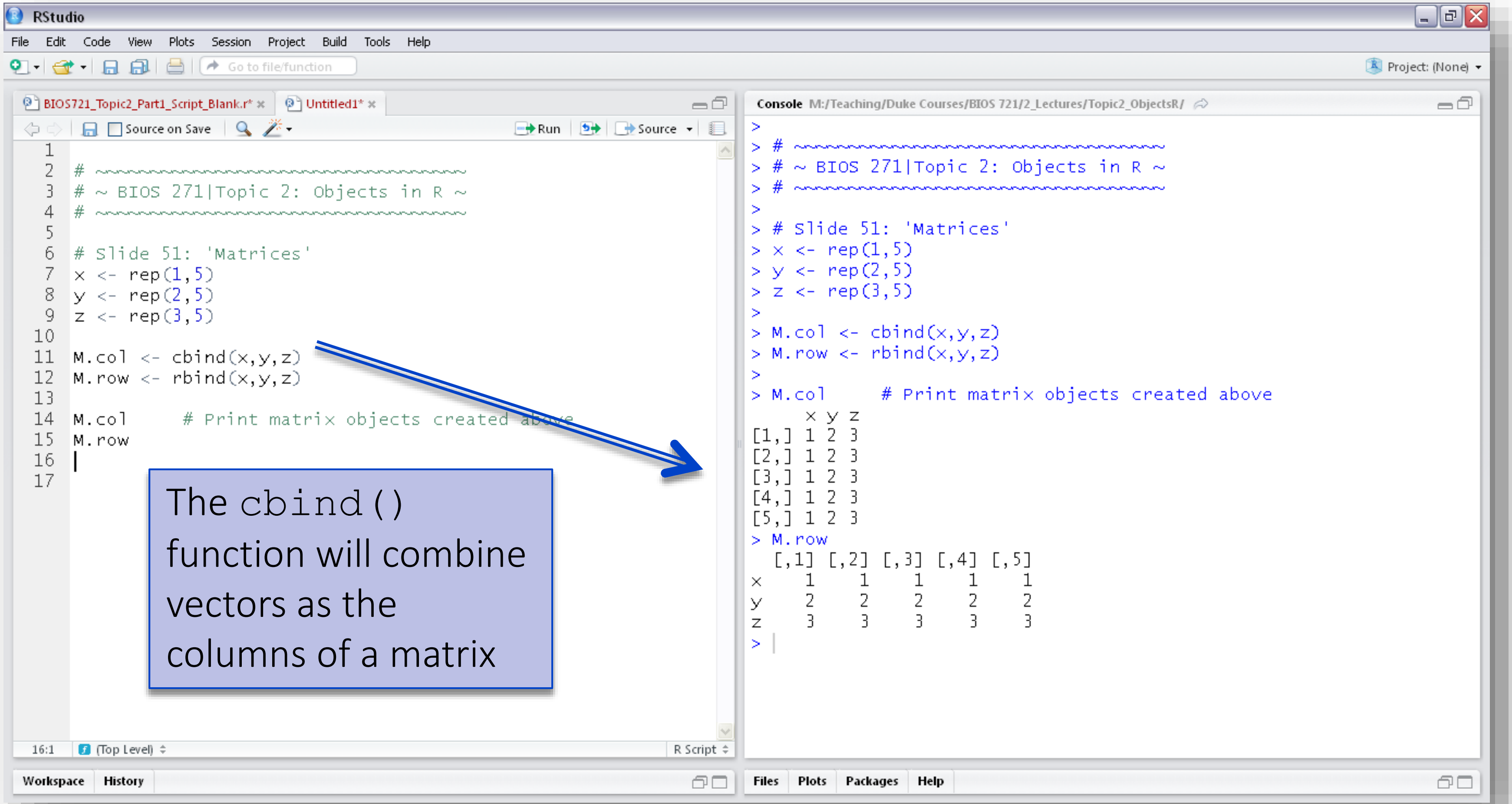


```r

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 51: 'Matrices'
x <- rep(1,5)
y <- rep(2,5)
z <- rep(3,5)

M.col <- cbind(x,y,z)
M.row <- rbind(x,y,z)

M.col     # Print matrix objects created above
M.row

```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 51: 'Matrices'
> x <- rep(1,5)
> y <- rep(2,5)
> z <- rep(3,5)
>
> M.col <- cbind(x,y,z)
> M.row <- rbind(x,y,z)
>
> M.col     # Print matrix objects created above
     x y z
[1,] 1 2 3
[2,] 1 2 3
[3,] 1 2 3
[4,] 1 2 3
[5,] 1 2 3
> M.row
  [,1] [,2] [,3] [,4] [,5]
x    1    1    1    1    1
y    2    2    2    2    2
z    3    3    3    3    3
>
```

The `rep()` function will create a vector by replicating the scalar value (numeric, character, or logical) provided – another great shortcut!

R Objects

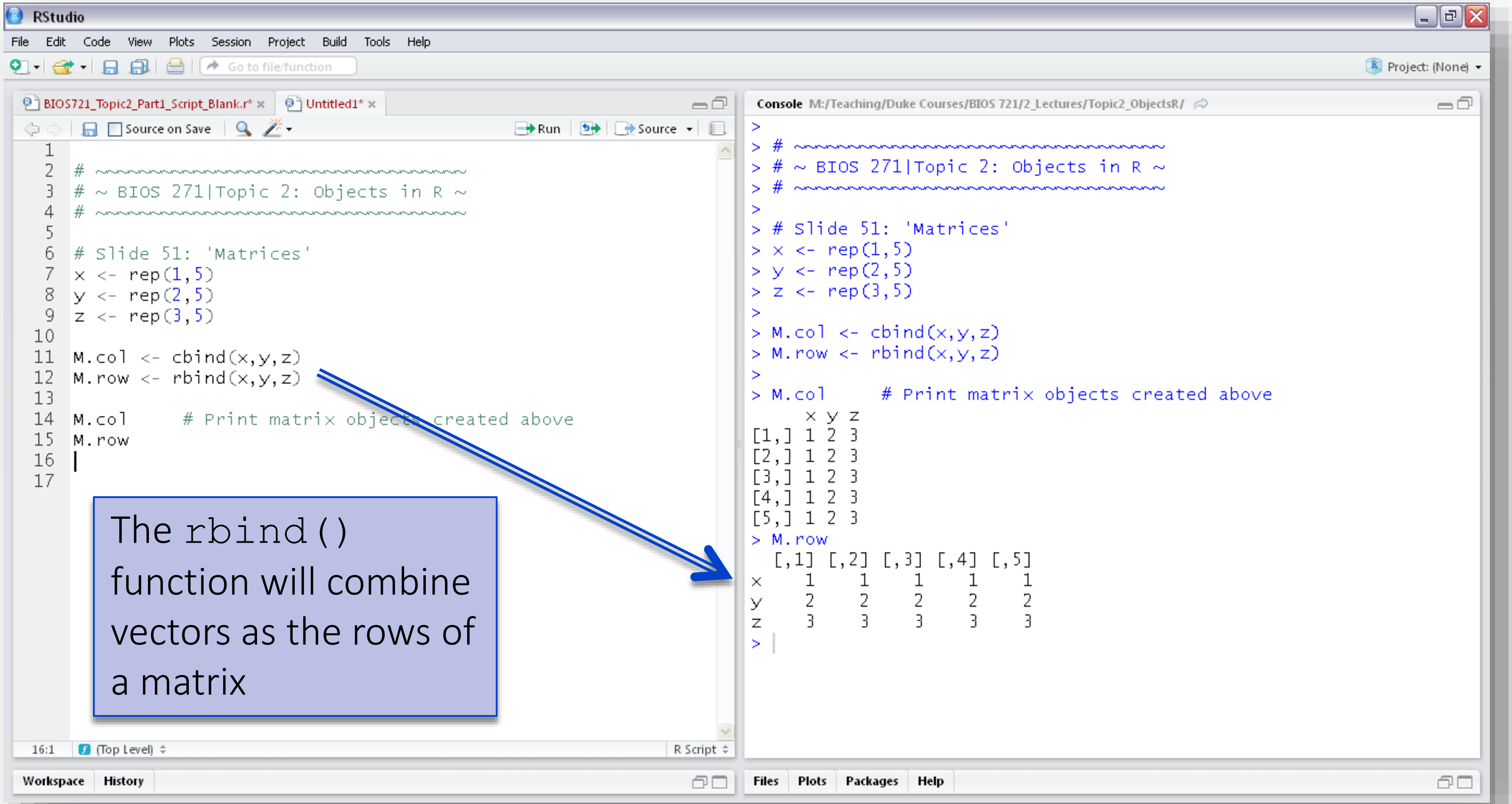


```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 51: 'Matrices'
x <- rep(1,5)
y <- rep(2,5)
z <- rep(3,5)

M.col <- cbind(x,y,z)
M.row <- rbind(x,y,z)

M.col     # Print matrix objects created above
M.row

```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 51: 'Matrices'
> x <- rep(1,5)
> y <- rep(2,5)
> z <- rep(3,5)
>
> M.col <- cbind(x,y,z)
> M.row <- rbind(x,y,z)
>
> M.col     # Print matrix objects created above
     x y z
[1,] 1 2 3
[2,] 1 2 3
[3,] 1 2 3
[4,] 1 2 3
[5,] 1 2 3
> M.row
  [,1] [,2] [,3] [,4] [,5]
x    1    1    1    1    1
y    2    2    2    2    2
z    3    3    3    3    3
>
```

The `cbind()` function will combine vectors as the columns of a matrix

R Objects



```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 51: 'Matrices'
x <- rep(1,5)
y <- rep(2,5)
z <- rep(3,5)

M.col <- cbind(x,y,z)
M.row <- rbind(x,y,z)

M.col     # Print matrix objects created above
M.row

```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 51: 'Matrices'
> x <- rep(1,5)
> y <- rep(2,5)
> z <- rep(3,5)
>
> M.col <- cbind(x,y,z)
> M.row <- rbind(x,y,z)
>
> M.col     # Print matrix objects created above
     x y z
[1,] 1 2 3
[2,] 1 2 3
[3,] 1 2 3
[4,] 1 2 3
[5,] 1 2 3
> M.row
  [,1] [,2] [,3] [,4] [,5]
x    1    1    1    1    1
y    2    2    2    2    2
z    3    3    3    3    3
>
```

The `rbind()` function will combine vectors as the rows of a matrix

R Objects

- Arrays (stacks of matrices)
  - Create using the R function ‘array’:

```
array(data,dim,...)
```

  - Examples
    - `A1 <- array(1:16,dim=c(4,2,2))`
    -
```
A2 <- array(c(rep(1,4),rep(2,4),rep(3,4)),
        dim=c(2,2,3),
        dimnames=list(c("R1","R2"),
        c("C1", "C2"),
        c("ones","twos","threes")))
```

R Objects

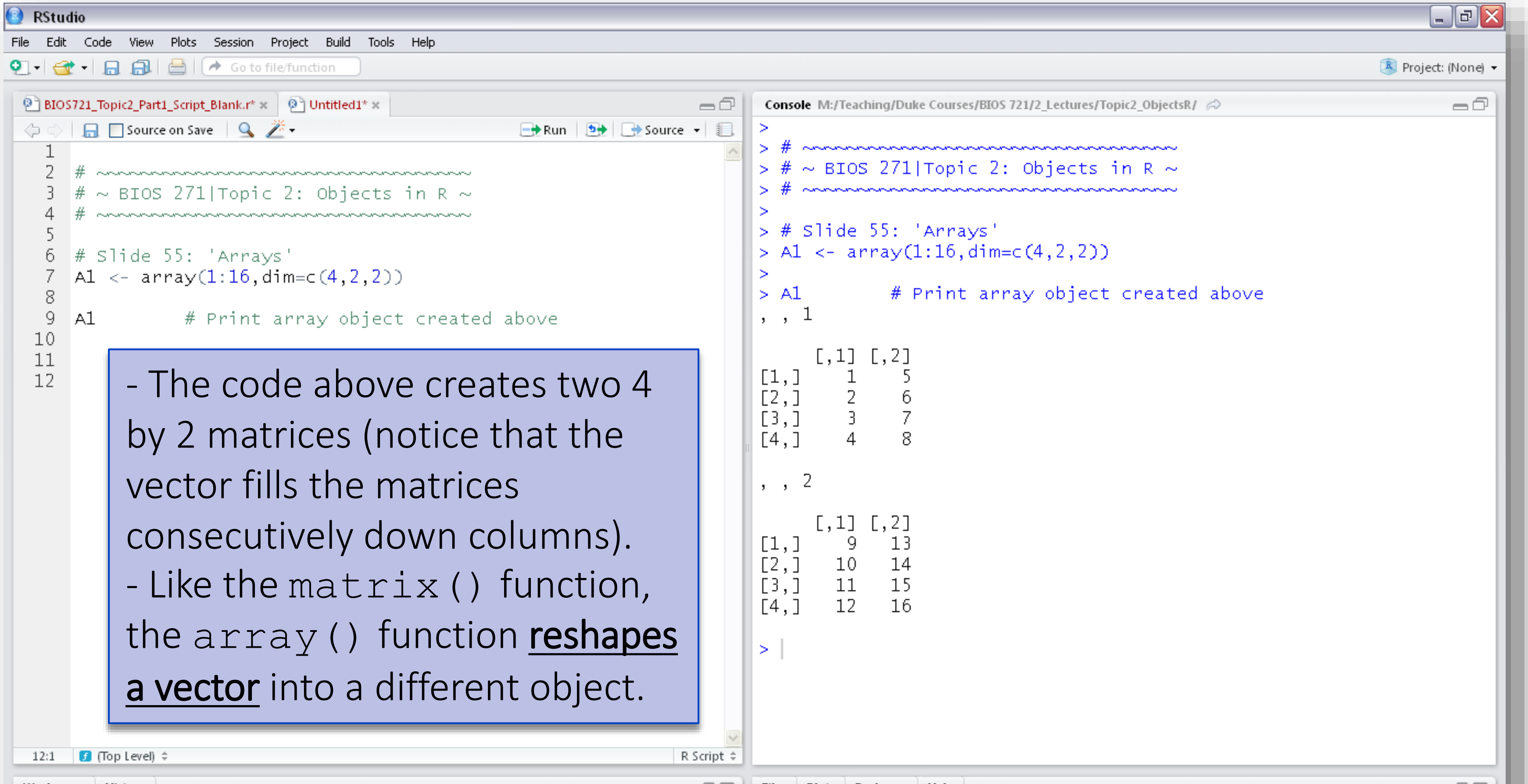
- Arrays (stacks of matrices)
  - Create using the R function 'array':

```
array(data,dim,...)
```

data = any VECTOR of values (numeric, character, or logical)
dim = a vector of length 3 giving the no. of rows in each matrix, no. of columns in each matrix, and the no. of matrices

  - Examples
    - `A1 <- array(1:16,dim=c(4,2,2))`
    -
```
A2 <- array(c(rep(1,4),rep(2,4),rep(3,4)),
  dim=c(2,2,3),dimnames=list(c("R1","R2"),
      c("C1", "C2"),
      c("ones","twos","threes")))
```

R Objects

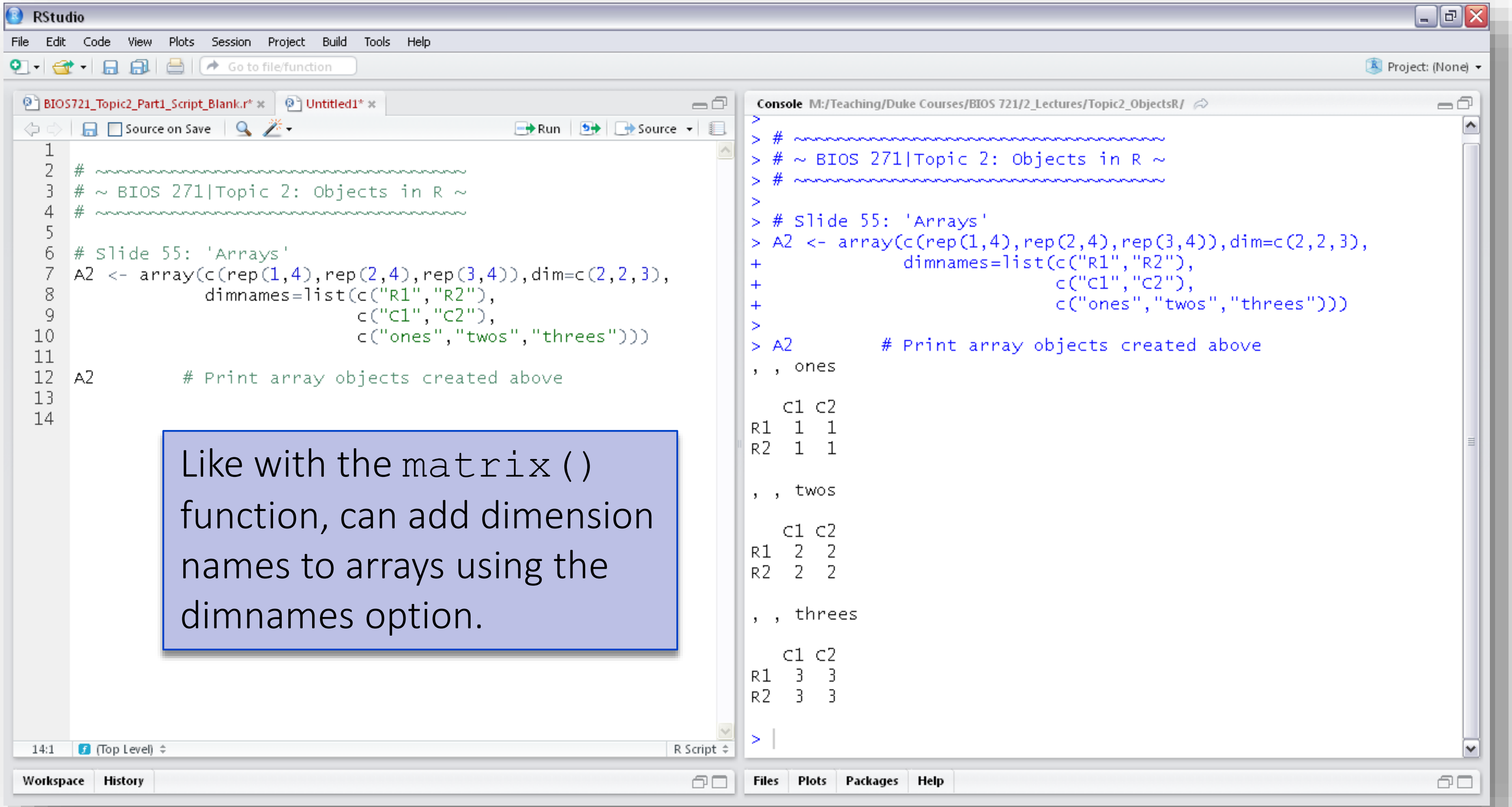


```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 55: 'Arrays'
A1 <- array(1:16,dim=c(4,2,2))

A1          # Print array object created above
```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 55: 'Arrays'
> A1 <- array(1:16,dim=c(4,2,2))
>
> A1          # Print array object created above
, , 1

     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8

, , 2

     [,1] [,2]
[1,]    9   13
[2,]   10   14
[3,]   11   15
[4,]   12   16

> 
```

- The code above creates two 4 by 2 matrices (notice that the vector fills the matrices consecutively down columns).
- Like the `matrix()` function, the `array()` function **reshapes a vector** into a different object.

R Objects



```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 55: 'Arrays'
A2 <- array(c(rep(1,4),rep(2,4),rep(3,4)),dim=c(2,2,3),
            dimnames=list(c("R1","R2"),
                          c("C1","C2"),
                          c("ones","twos","threes")))

A2          # Print array objects created above
```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 55: 'Arrays'
> A2 <- array(c(rep(1,4),rep(2,4),rep(3,4)),dim=c(2,2,3),
+             dimnames=list(c("R1","R2"),
+                           c("C1","C2"),
+                           c("ones","twos","threes")))
>
> A2          # Print array objects created above
, , ones

   C1 C2
R1  1  1
R2  1  1

, , twos

   C1 C2
R1  2  2
R2  2  2

, , threes

   C1 C2
R1  3  3
R2  3  3

>
```

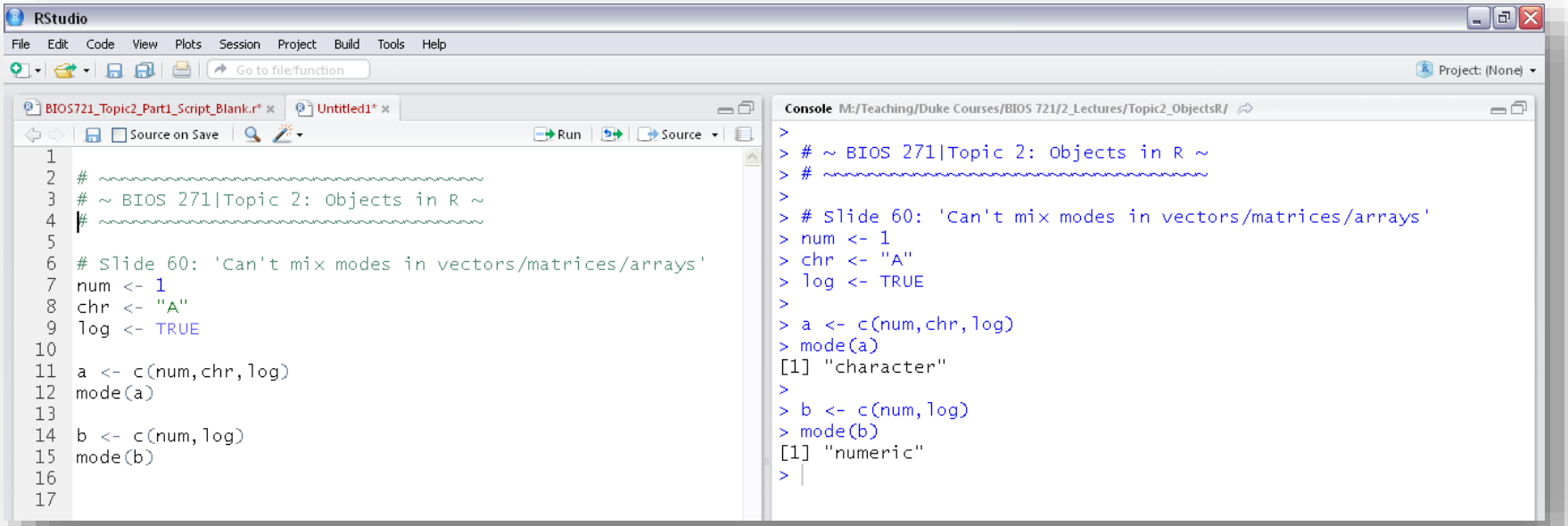
Like with the `matrix()` function, can add dimension names to arrays using the dimnames option.

R Objects

- For all of the R objects that we have talked about so far, all elements have to be of the SAME mode!
  - That is, the following objects have to have all numeric, character, or logical elements:
    - Vectors
    - Matrices
    - Arrays
  - No mixing!

R Objects

- If there is mixing, R will coerce the elements of these objects to all be of the same mode.



```r

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 60: 'Can't mix modes in vectors/matrices/arrays'
num <- 1
chr <- "A"
log <- TRUE

a <- c(num,chr,log)
mode(a)

b <- c(num,log)
mode(b)


```

```
>
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 60: 'Can't mix modes in vectors/matrices/arrays'
> num <- 1
> chr <- "A"
> log <- TRUE
>
> a <- c(num,chr,log)
> mode(a)
[1] "character"
>
> b <- c(num,log)
> mode(b)
[1] "numeric"
> 
```

R Objects

- But in consulting and research that is not always practical or feasible.
  - Most data sets will have a mixture of (at least) numeric and character data values.
  - The last two R data objects allow you to mix modes or even object types.
    - Data Frames
    - Lists

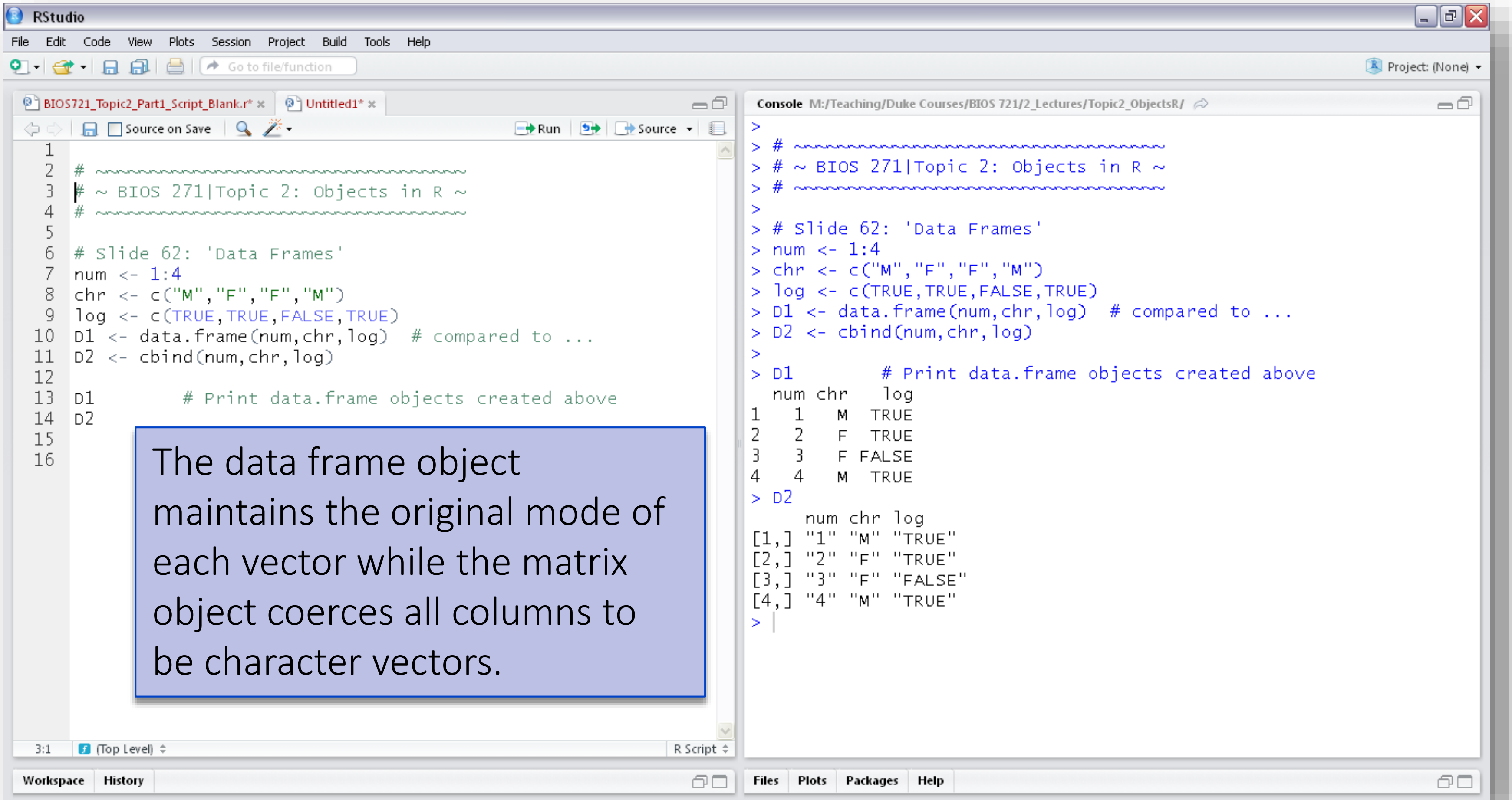
R Objects

- Data Frames
  - Allow you to combine vectors of different modes together to create a data set (essentially a matrix)
  - Create using the R function 'data.frame':

```
data.frame(data,...)
```

  - Example
    - `num <- 1:4`
    - `chr <- c("M","F","F","M")`
    - `log <- c(TRUE,TRUE,FALSE,TRUE)`
    - `D1 <- data.frame(num,chr,log)` compare to
    - `D2 <- cbind(num,chr,log)`

R Objects



```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 62: 'Data Frames'
num <- 1:4
chr <- c("M","F","F","M")
log <- c(TRUE,TRUE,FALSE,TRUE)
D1 <- data.frame(num,chr,log)  # compared to ...
D2 <- cbind(num,chr,log)

D1        # Print data.frame objects created above
D2


```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 62: 'Data Frames'
> num <- 1:4
> chr <- c("M","F","F","M")
> log <- c(TRUE,TRUE,FALSE,TRUE)
> D1 <- data.frame(num,chr,log)  # compared to ...
> D2 <- cbind(num,chr,log)
>
> D1        # Print data.frame objects created above
  num chr   log
1   1   M  TRUE
2   2   F  TRUE
3   3   F FALSE
4   4   M  TRUE
> D2
     num chr log
[1,] "1" "M" "TRUE"
[2,] "2" "F" "TRUE"
[3,] "3" "F" "FALSE"
[4,] "4" "M" "TRUE"
> 
```

The data frame object maintains the original mode of each vector while the matrix object coerces all columns to be character vectors.

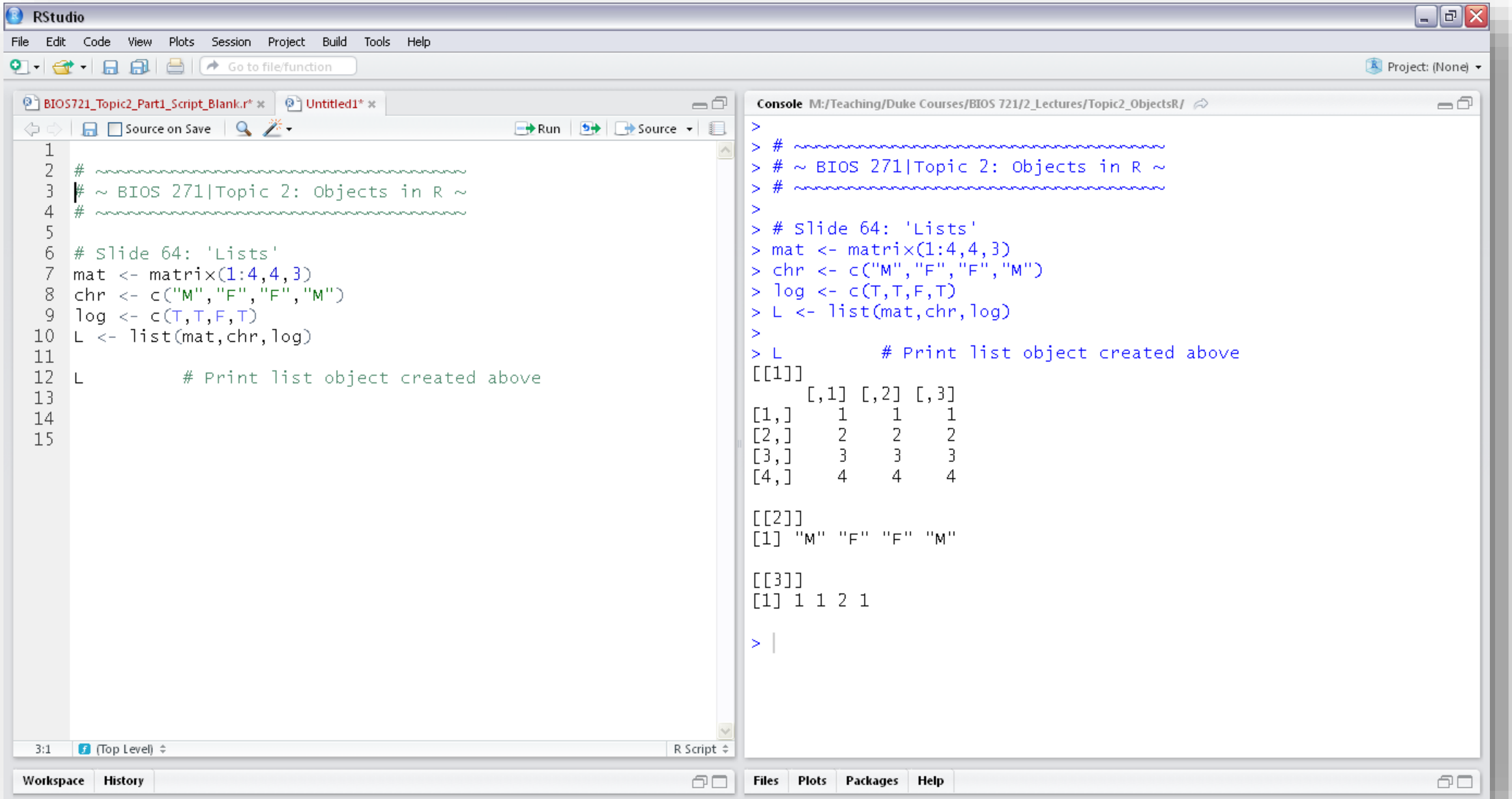
R Objects

- Lists
  - Allow you to combine objects of different modes and objects of different classes (types) into a single object
  - Create using the R function 'list':

```
list(data,...)
```

  - Example
    - `mat <- matrix(1:4,4,3)`
    - `chr <- c("M","F","F","M")`
    - `log <- c(TRUE,TRUE,FALSE,TRUE)`
    - `L <- list(mat,chr,log)`

R Objects



```

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~ BIOS 271|Topic 2: Objects in R ~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Slide 64: 'Lists'
mat <- matrix(1:4,4,3)
chr <- c("M","F","F","M")
log <- c(T,T,F,T)
L <- list(mat,chr,log)

L          # Print list object created above
```

```
>
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> # ~ BIOS 271|Topic 2: Objects in R ~
> # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
>
> # Slide 64: 'Lists'
> mat <- matrix(1:4,4,3)
> chr <- c("M","F","F","M")
> log <- c(T,T,F,T)
> L <- list(mat,chr,log)
>
> L          # Print list object created above
[[1]]
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
[4,]    4    4    4

[[2]]
[1] "M" "F" "F" "M"

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

>
```

R Objects

- If you want to know the mode of the elements in an object (that does not allow mixing) you can use
  - `mode(object)`
  - Returns one of the following
    - `"numeric"`
    - `"character"`
    - `"logical"`

R Objects

- If you want to know the type of object that you are dealing with you can use
  - `class(object)`
  - Returns one of the following
    - `"numeric"`
    - `"integer"`
    - `"character"`
    - `"logical"`
    - `"matrix"`
    - `"array"`
    - `"data.frame"`
    - `"list"`

a.k.a. scalar/vector of a particular mode

For Next Time

- Next Time Topic 1 continued (Part 2)
  - Operations performed on objects