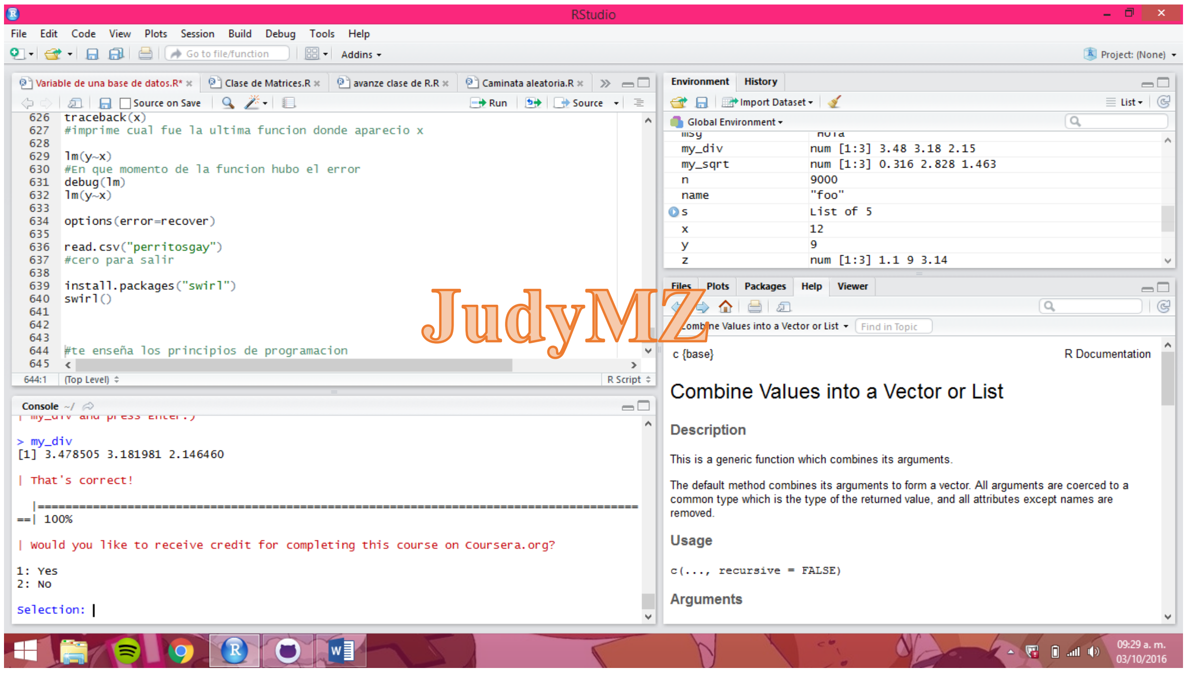
# 1: Basic Building Blocks



```r
traceback(x)
#imprime cual fue la ultima funcion donde aparecio x

lm(y~x)
#En que momento de la funcion hubo el error
debug(lm)
lm(y~x)

options(error=recover)

read.csv("perritosgay")
#cero para salir

install.packages("swirl")
swirl()



#te enseña los principios de programacion
```

```
my_div          num [1:3] 3.48 3.18 2.15
my_sqrt         num [1:3] 0.316 2.828 1.463
n               9000
name            "foo"
s               List of 5
x               12
y               9
z               num [1:3] 1.1 9 3.14
```

```
> my_div
[1] 3.478505 3.181981 2.146460

| That's correct!

  |==========================================================================
==| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No

Selection: |
```

c {base} R Documentation

Combine Values into a Vector or List

Description

This is a generic function which combines its arguments.

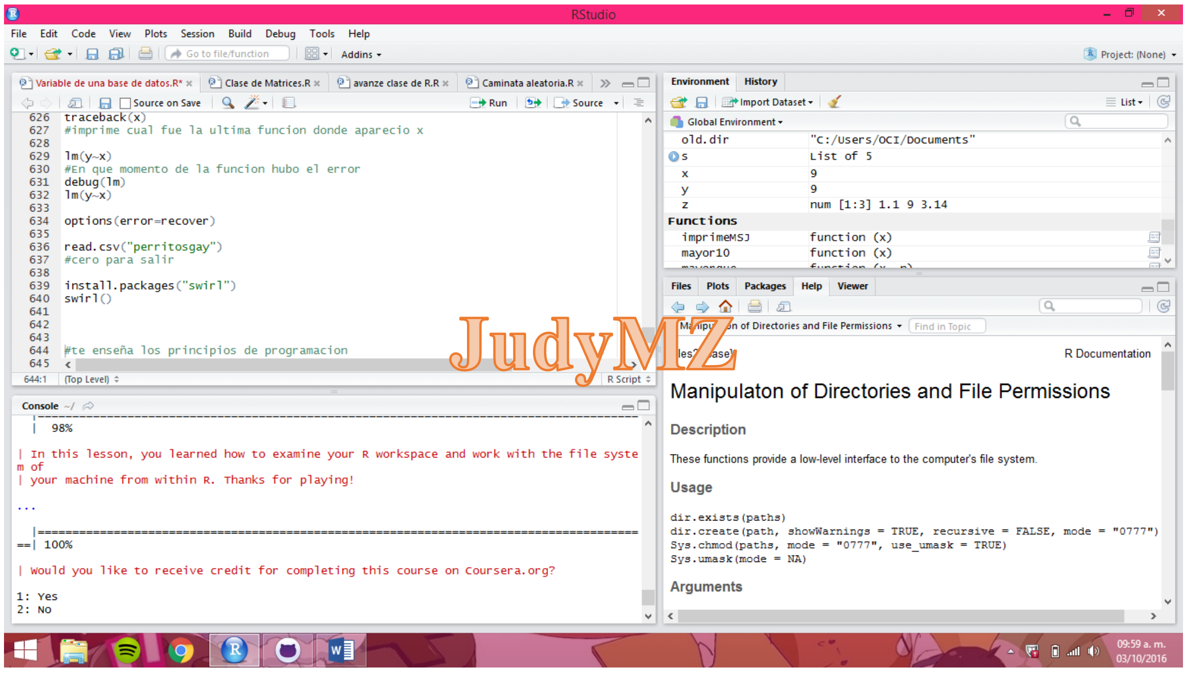
The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed.

Usage

```
c(..., recursive = FALSE)
```

Arguments

# 2: Workspace and Files



```r
traceback(x)
#imprime cual fue la ultima funcion donde aparecio x

lm(y~x)
#En que momento de la funcion hubo el error
debug(lm)
lm(y~x)

options(error=recover)

read.csv("perritosgay")
#cero para salir

install.packages("swirl")
swirl()



#te enseña los principios de programacion
```

```
old.dir         "C:/Users/OCI/Documents"
s               List of 5
x               9
y               9
z               num [1:3] 1.1 9 3.14
Functions
imprimeMSJ      function (x)
mayor10         function (x)
```

```
  |   98%

| In this lesson, you learned how to examine your R workspace and work with the file syste
m of
| your machine from within R. Thanks for playing!

...

  |==========================================================================
==| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No
```

R Documentation

Manipulaton of Directories and File Permissions

Description

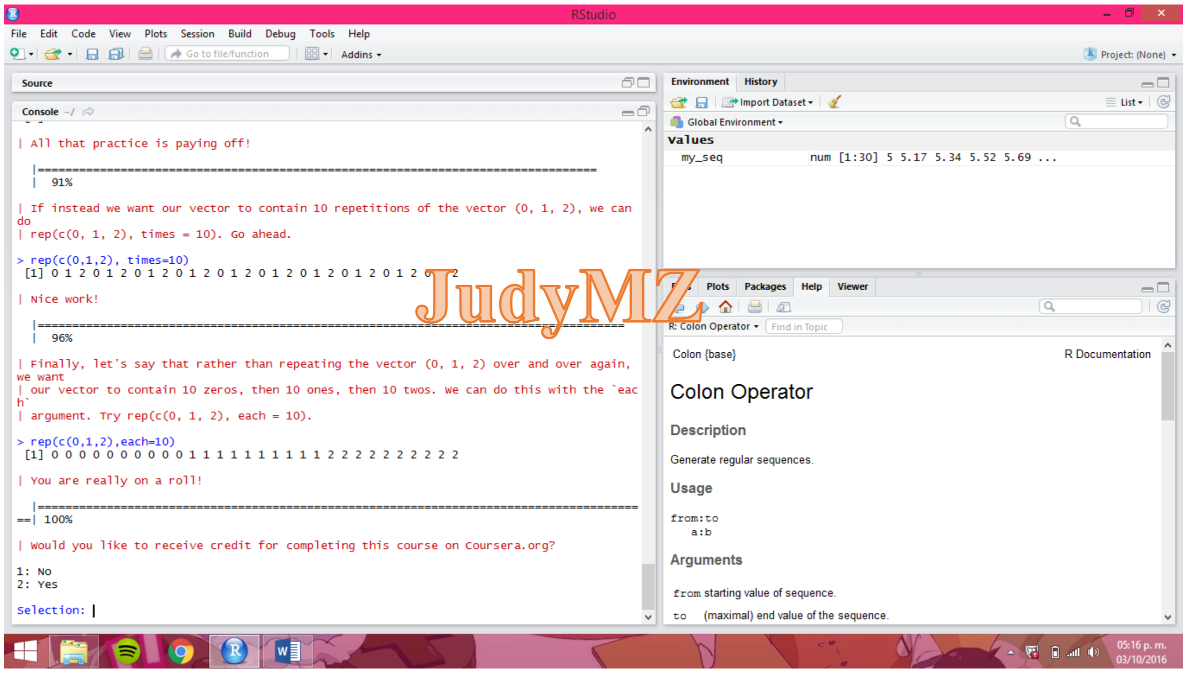
These functions provide a low-level interface to the computer's file system.

Usage

```
dir.exists(paths)
dir.create(path, showWarnings = TRUE, recursive = FALSE, mode = "0777")
Sys.chmod(paths, mode = "0777", use_umask = TRUE)
Sys.umask(mode = NA)
```

Arguments

# 3: Sequences of Numbers



```
| All that practice is paying off!

  |=============================================================================
  |  91%

| If instead we want our vector to contain 10 repetitions of the vector (0, 1, 2), we can
do
| rep(c(0, 1, 2), times = 10). Go ahead.

> rep(c(0,1,2), times=10)
 [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 [illegible] 2

| Nice work!

  |=================================================================================
  |  96%

| Finally, let's say that rather than repeating the vector (0, 1, 2) over and over again,
we want
| our vector to contain 10 zeros, then 10 ones, then 10 twos. We can do this with the `eac
h`
| argument. Try rep(c(0, 1, 2), each = 10).

> rep(c(0,1,2),each=10)
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2

| You are really on a roll!

  |======================================================================================
==| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: No
2: Yes

Selection: |
```

```
Values
  my_seq          num [1:30] 5 5.17 5.34 5.52 5.69 ...
```

Colon {base} R Documentation

Colon Operator

Description

Generate regular sequences.

Usage

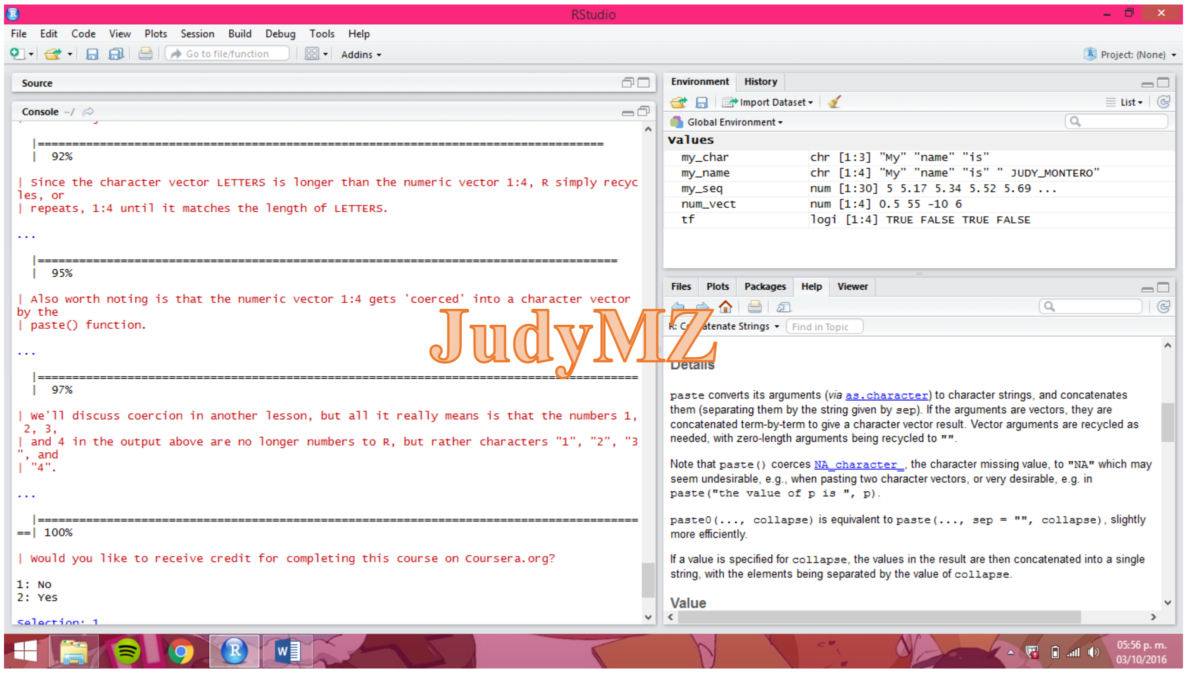
```
from:to
   a:b
```

Arguments

`from` starting value of sequence.

`to` (maximal) end value of the sequence.

# 4: Vectors



```
  |============================================================================
  |  92%

| Since the character vector LETTERS is longer than the numeric vector 1:4, R simply recyc
les, or
| repeats, 1:4 until it matches the length of LETTERS.

...

  |==============================================================================
  |  95%

| Also worth noting is that the numeric vector 1:4 gets 'coerced' into a character vector
by the
| paste() function.

...

  |=================================================================================
  |  97%

| We'll discuss coercion in another lesson, but all it really means is that the numbers 1,
 2, 3,
| and 4 in the output above are no longer numbers to R, but rather characters "1", "2", "3
", and
| "4".

...

  |==================================================================================
==| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: No
2: Yes

Selection: 1
```

```
Values
  my_char         chr [1:3] "My" "name" "is"
  my_name         chr [1:4] "My" "name" "is" " JUDY_MONTERO"
  my_seq          num [1:30] 5 5.17 5.34 5.52 5.69 ...
  num_vect        num [1:4] 0.5 55 -10 6
  tf              logi [1:4] TRUE FALSE TRUE FALSE
```

Details

`paste` converts its arguments (*via* `as.character`) to character strings, and concatenates them (separating them by the string given by `sep`). If the arguments are vectors, they are concatenated term-by-term to give a character vector result. Vector arguments are recycled as needed, with zero-length arguments being recycled to `""`.

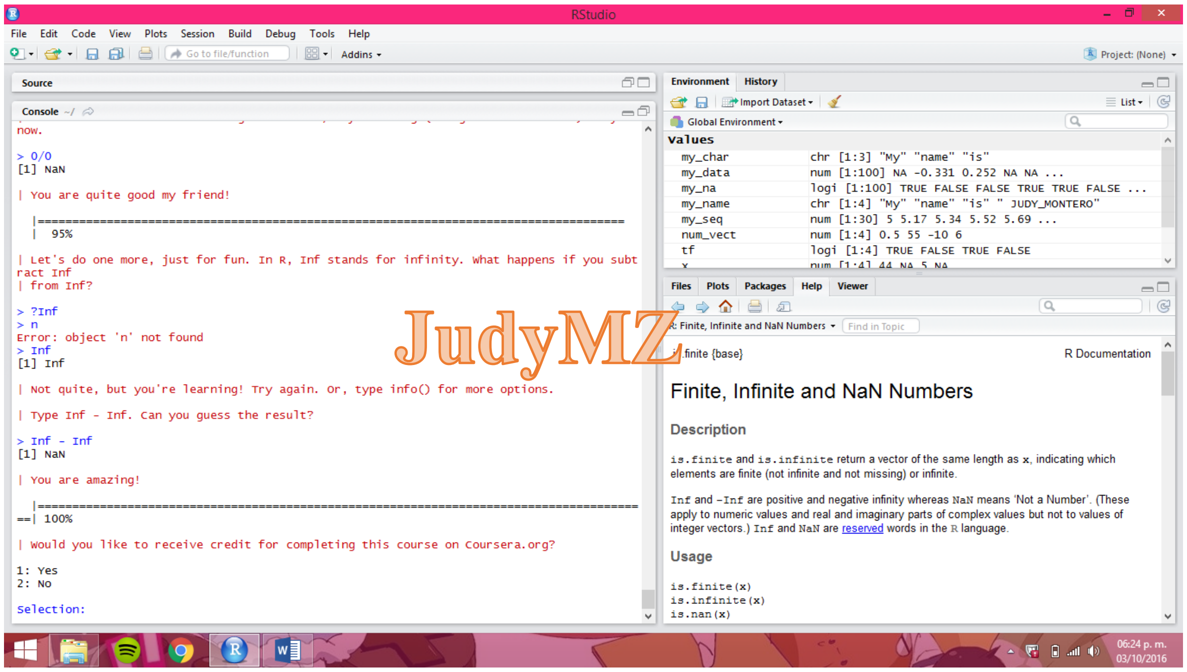
Note that `paste()` coerces `NA_character_`, the character missing value, to `"NA"` which may seem undesirable, e.g., when pasting two character vectors, or very desirable, e.g. in `paste("the value of p is ", p)`.

`paste0(..., collapse)` is equivalent to `paste(..., sep = "", collapse)`, slightly more efficiently.

If a value is specified for `collapse`, the values in the result are then concatenated into a single string, with the elements being separated by the value of `collapse`.

Value

## 5: Missing Values



```
now.

> 0/0
[1] NaN

| You are quite good my friend!

  |=====================================================================================
  |  95%

| Let's do one more, just for fun. In R, Inf stands for infinity. What happens if you subt
ract Inf
| from Inf?

> ?Inf
> n
Error: object 'n' not found
> Inf
[1] Inf

| Not quite, but you're learning! Try again. Or, type info() for more options.

| Type Inf - Inf. Can you guess the result?

> Inf - Inf
[1] NaN

| You are amazing!

  |=========================================================================================
==| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No

Selection:
```

```
Values
  my_char      chr [1:3] "My" "name" "is"
  my_data      num [1:100] NA -0.331 0.252 NA NA ...
  my_na        logi [1:100] TRUE FALSE FALSE TRUE TRUE FALSE ...
  my_name      chr [1:4] "My" "name" "is" " JUDY_MONTERO"
  my_seq       num [1:30] 5 5.17 5.34 5.52 5.69 ...
  num_vect     num [1:4] 0.5 55 -10 6
  tf           logi [1:4] TRUE FALSE TRUE FALSE
  x            num [1:4] 44 NA 5 NA
```

is.finite {base} R Documentation

Finite, Infinite and NaN Numbers

Description

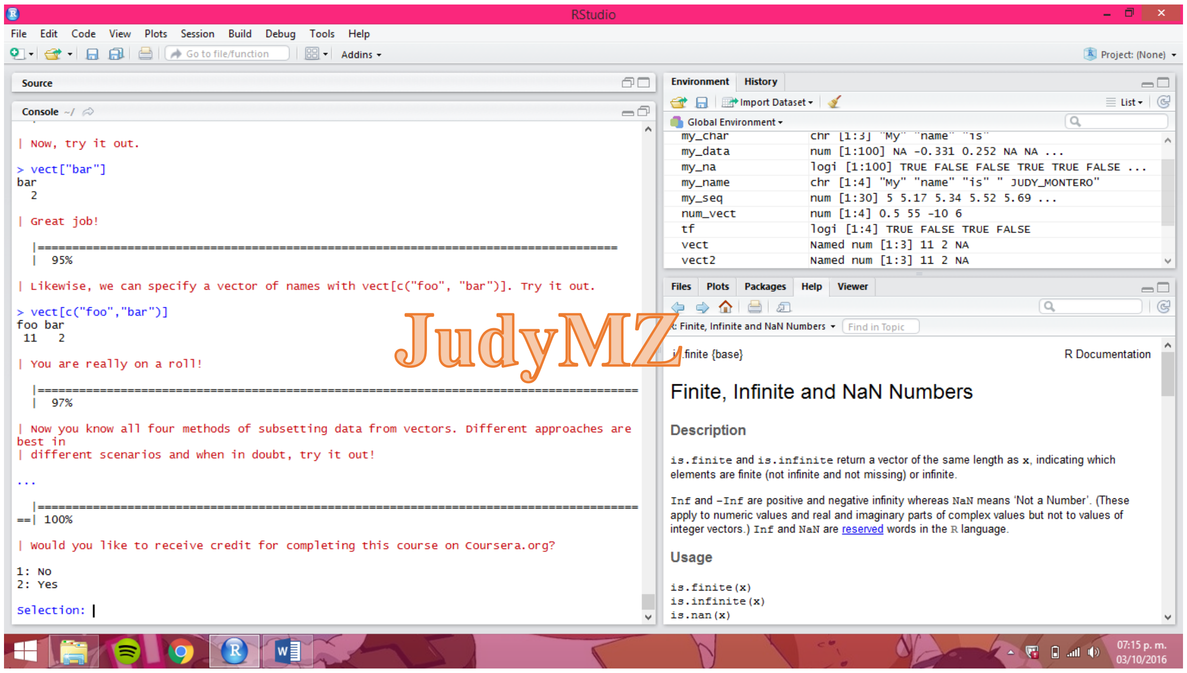
`is.finite` and `is.infinite` return a vector of the same length as `x`, indicating which elements are finite (not infinite and not missing) or infinite.

`Inf` and `-Inf` are positive and negative infinity whereas `NaN` means 'Not a Number'. (These apply to numeric values and real and imaginary parts of complex values but not to values of integer vectors.) `Inf` and `NaN` are reserved words in the R language.

Usage

```
is.finite(x)
is.infinite(x)
is.nan(x)
```

## 6: Subsetting Vectors



```
| Now, try it out.

> vect["bar"]
bar
  2

| Great job!

  |=====================================================================================
  |  95%

| Likewise, we can specify a vector of names with vect[c("foo", "bar")]. Try it out.

> vect[c("foo","bar")]
foo bar
 11   2

| You are really on a roll!

  |=========================================================================================
  |  97%

| Now you know all four methods of subsetting data from vectors. Different approaches are
best in
| different scenarios and when in doubt, try it out!

...

  |=========================================================================================
==| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: No
2: Yes

Selection: |
```

```
  my_char      chr [1:3] "My" "name" "is"
  my_data      num [1:100] NA -0.331 0.252 NA NA ...
  my_na        logi [1:100] TRUE FALSE FALSE TRUE TRUE FALSE ...
  my_name      chr [1:4] "My" "name" "is" " JUDY_MONTERO"
  my_seq       num [1:30] 5 5.17 5.34 5.52 5.69 ...
  num_vect     num [1:4] 0.5 55 -10 6
  tf           logi [1:4] TRUE FALSE TRUE FALSE
  vect         Named num [1:3] 11 2 NA
  vect2        Named num [1:3] 11 2 NA
```

is.finite {base} R Documentation

Finite, Infinite and NaN Numbers

Description

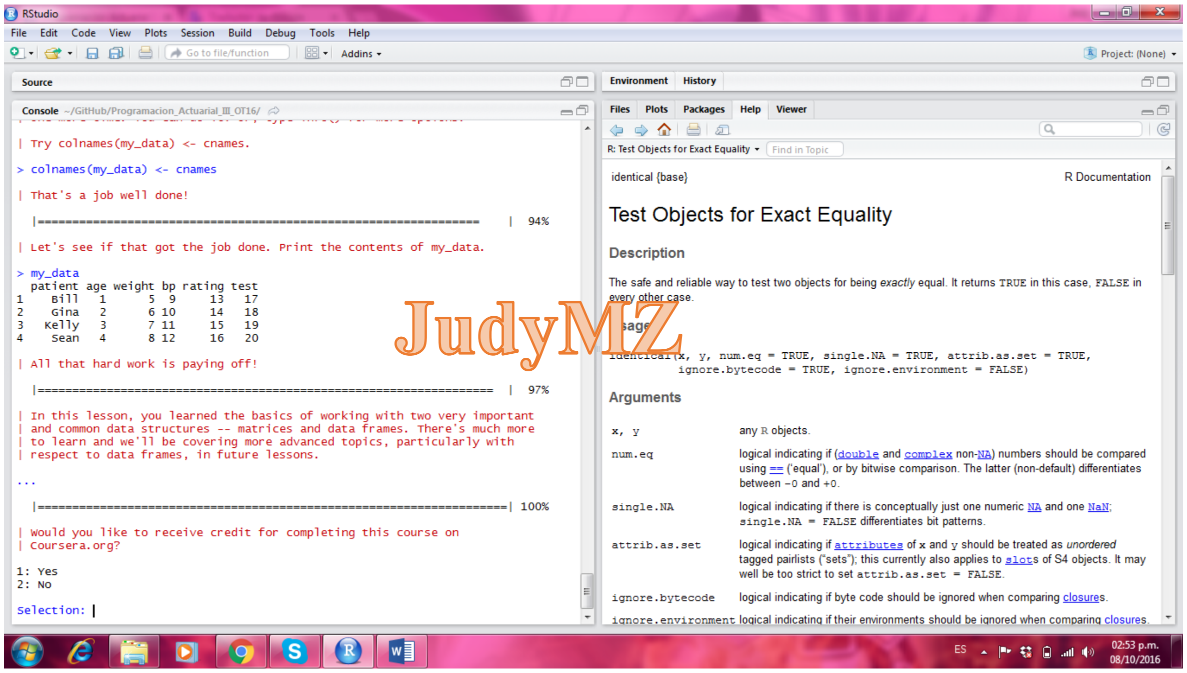
`is.finite` and `is.infinite` return a vector of the same length as `x`, indicating which elements are finite (not infinite and not missing) or infinite.

`Inf` and `-Inf` are positive and negative infinity whereas `NaN` means 'Not a Number'. (These apply to numeric values and real and imaginary parts of complex values but not to values of integer vectors.) `Inf` and `NaN` are reserved words in the R language.

Usage

```
is.finite(x)
is.infinite(x)
is.nan(x)
```

## 7: Matrices and Data Frames


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
Project: (None)
Source
Console ~/GitHub/Programacion_Actuarial_III_OT16/
Environment History
Files Plots Packages Help Viewer
R: Test Objects for Exact Equality
Find in Topic
ES
02:53 p.m.
08/10/2016

```
| Try colnames(my_data) <- cnames.

> colnames(my_data) <- cnames

| That's a job well done!

  |=================================================================     |  94%

| Let's see if that got the job done. Print the contents of my_data.

> my_data
  patient age weight bp rating test
1    Bill   1      5  9     13   17
2    Gina   2      6 10     14   18
3   Kelly   3      7 11     15   19
4    Sean   4      8 12     16   20

| All that hard work is paying off!

  |===================================================================   |  97%

| In this lesson, you learned the basics of working with two very important
| and common data structures -- matrices and data frames. There's much more
| to learn and we'll be covering more advanced topics, particularly with
| respect to data frames, in future lessons.

...

  |======================================================================| 100%

| Would you like to receive credit for completing this course on
| Coursera.org?

1: Yes
2: No

Selection: |
```

identical {base}  R Documentation

### Test Objects for Exact Equality

**Description**

The safe and reliable way to test two objects for being *exactly* equal. It returns `TRUE` in this case, `FALSE` in every other case.

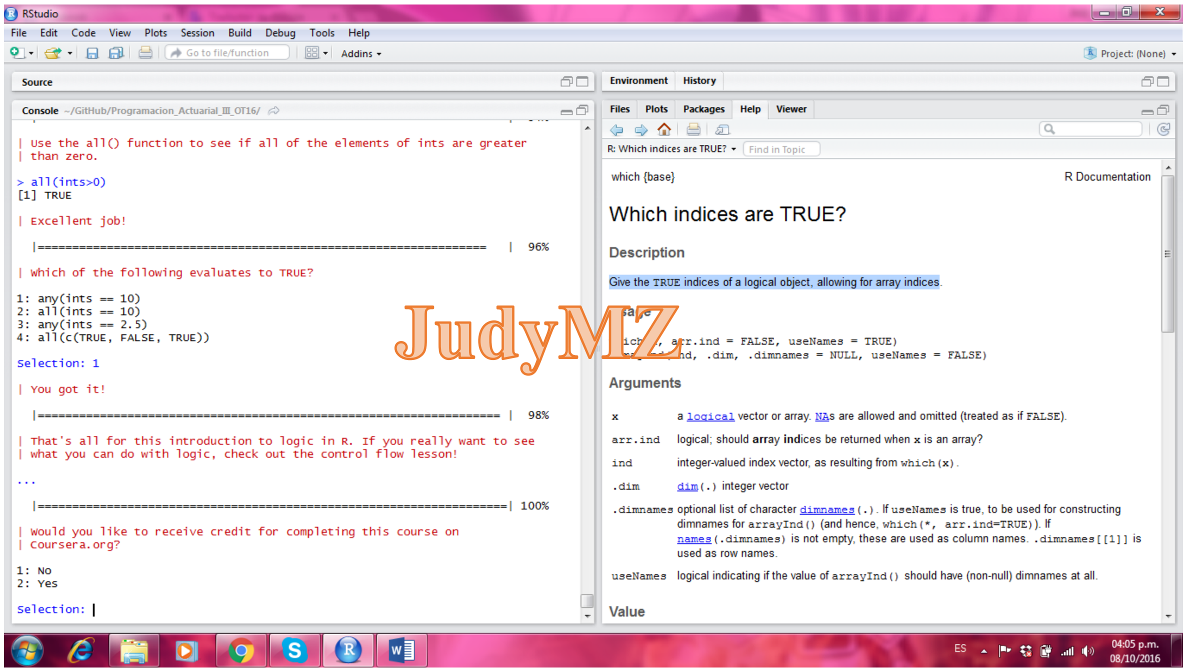
**Usage**

```
identical(x, y, num.eq = TRUE, single.NA = TRUE, attrib.as.set = TRUE,
          ignore.bytecode = TRUE, ignore.environment = FALSE)
```

**Arguments**

| | |
|---|---|
| `x, y` | any R objects. |
| `num.eq` | logical indicating if (`double` and `complex` non-`NA`) numbers should be compared using `==` ('equal'), or by bitwise comparison. The latter (non-default) differentiates between `-0` and `+0`. |
| `single.NA` | logical indicating if there is conceptually just one numeric `NA` and one `NaN`; `single.NA = FALSE` differentiates bit patterns. |
| `attrib.as.set` | logical indicating if `attributes` of `x` and `y` should be treated as *unordered* tagged pairlists ("sets"); this currently also applies to `slot`s of S4 objects. It may well be too strict to set `attrib.as.set = FALSE`. |
| `ignore.bytecode` | logical indicating if byte code should be ignored when comparing closures. |
| `ignore.environment` | logical indicating if their environments should be ignored when comparing closures. |

## 8: Logic


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
Project: (None)
Source
Console ~/GitHub/Programacion_Actuarial_III_OT16/
Environment History
Files Plots Packages Help Viewer
R: Which indices are TRUE?
Find in Topic
ES
04:05 p.m.
08/10/2016

```
| Use the all() function to see if all of the elements of ints are greater
| than zero.

> all(ints>0)
[1] TRUE

| Excellent job!

  |==================================================================    |  96%

| Which of the following evaluates to TRUE?

1: any(ints == 10)
2: all(ints == 10)
3: any(ints == 2.5)
4: all(c(TRUE, FALSE, TRUE))

Selection: 1

| You got it!

  |====================================================================  |  98%

| That's all for this introduction to logic in R. If you really want to see
| what you can do with logic, check out the control flow lesson!

...

  |======================================================================| 100%

| Would you like to receive credit for completing this course on
| Coursera.org?

1: No
2: Yes

Selection: |
```

which {base}  R Documentation

### Which indices are TRUE?

**Description**

Give the `TRUE` indices of a logical object, allowing for array indices.

**Usage**

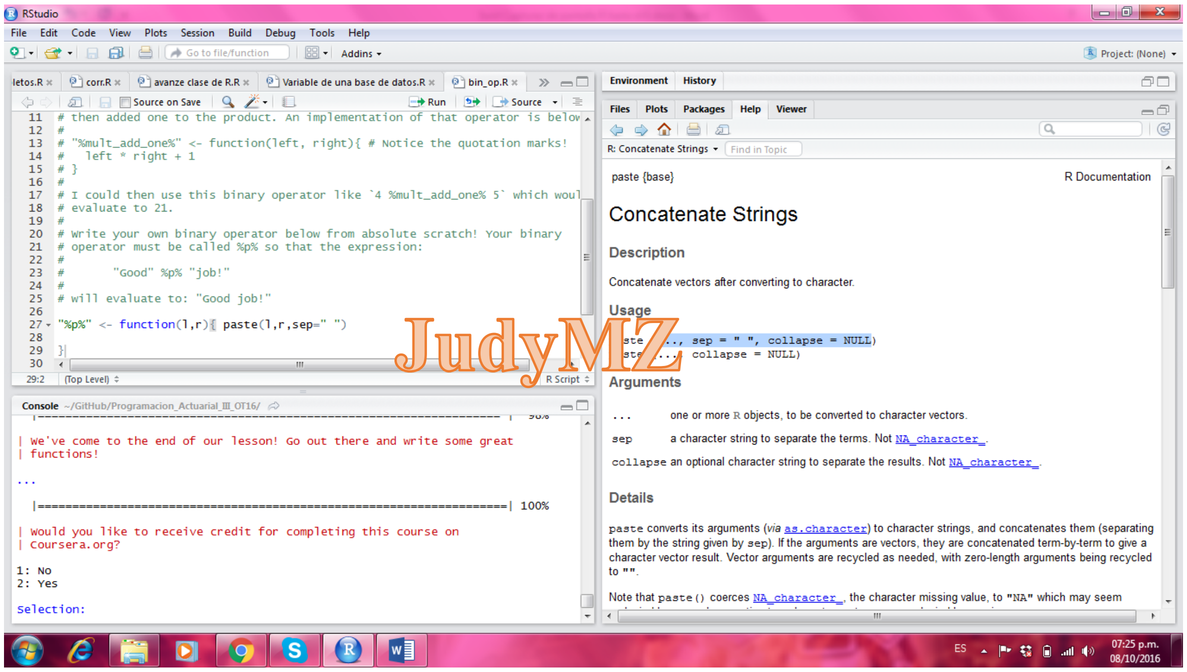
```
which(x, arr.ind = FALSE, useNames = TRUE)
arrayInd(ind, .dim, .dimnames = NULL, useNames = FALSE)
```

**Arguments**

| | |
|---|---|
| `x` | a `logical` vector or array. `NA`s are allowed and omitted (treated as if `FALSE`). |
| `arr.ind` | logical; should **arr**ay **ind**ices be returned when `x` is an array? |
| `ind` | integer-valued index vector, as resulting from `which(x)`. |
| `.dim` | `dim(.)` integer vector |
| `.dimnames` | optional list of character `dimnames(.)`. If `useNames` is true, to be used for constructing dimnames for `arrayInd()` (and hence, `which(*, arr.ind=TRUE)`). If `names(.dimnames)` is not empty, these are used as column names. `.dimnames[[1]]` is used as row names. |
| `useNames` | logical indicating if the value of `arrayInd()` should have (non-null) dimnames at all. |

**Value**

## 9: Functions



```
# then added one to the product. An implementation of that operator is below
#
# "%mult_add_one%" <- function(left, right){ # Notice the quotation marks!
#   left * right + 1
# }
#
# I could then use this binary operator like `4 %mult_add_one% 5` which woul
# evaluate to 21.
#
# Write your own binary operator below from absolute scratch! Your binary
# operator must be called %p% so that the expression:
#
#       "Good" %p% "job!"
#
# will evaluate to: "Good job!"

"%p%" <- function(l,r){ paste(l,r,sep=" ")

}
```

```
| We've come to the end of our lesson! Go out there and write some great
| functions!

...

  |=====================================================================| 100%

| Would you like to receive credit for completing this course on
| Coursera.org?

1: No
2: Yes

Selection:
```

paste {base} R Documentation

Concatenate Strings

Description

Concatenate vectors after converting to character.

Usage

```
paste (..., sep = " ", collapse = NULL)
paste0(..., collapse = NULL)
```

Arguments

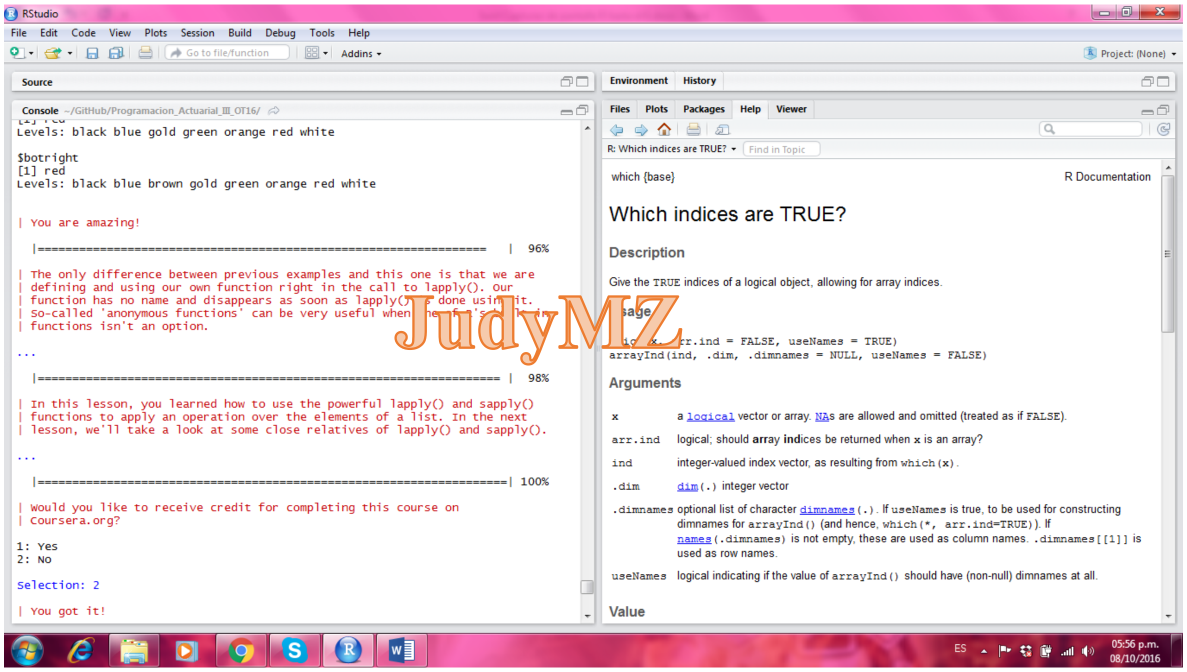
| | |
|---|---|
| `...` | one or more R objects, to be converted to character vectors. |
| `sep` | a character string to separate the terms. Not `NA_character_`. |
| `collapse` | an optional character string to separate the results. Not `NA_character_`. |

Details

`paste` converts its arguments (*via* `as.character`) to character strings, and concatenates them (separating them by the string given by `sep`). If the arguments are vectors, they are concatenated term-by-term to give a character vector result. Vector arguments are recycled as needed, with zero-length arguments being recycled to `""`.

Note that `paste()` coerces `NA_character_`, the character missing value, to `"NA"` which may seem

## 10: lapply and sapply



```
Levels: black blue gold green orange red white

$botright
[1] red
Levels: black blue brown gold green orange red white


| You are amazing!

  |=================================================================   |  96%

| The only difference between previous examples and this one is that we are
| defining and using our own function right in the call to lapply(). Our
| function has no name and disappears as soon as lapply() is done using it.
| So-called 'anonymous functions' can be very useful when one of R's built-in
| functions isn't an option.

...

  |====================================================================   |  98%

| In this lesson, you learned how to use the powerful lapply() and sapply()
| functions to apply an operation over the elements of a list. In the next
| lesson, we'll take a look at some close relatives of lapply() and sapply().

...

  |=====================================================================| 100%

| Would you like to receive credit for completing this course on
| Coursera.org?

1: Yes
2: No

Selection: 2

| You got it!
```

which {base} R Documentation

Which indices are TRUE?

Description

Give the `TRUE` indices of a logical object, allowing for array indices.

Usage

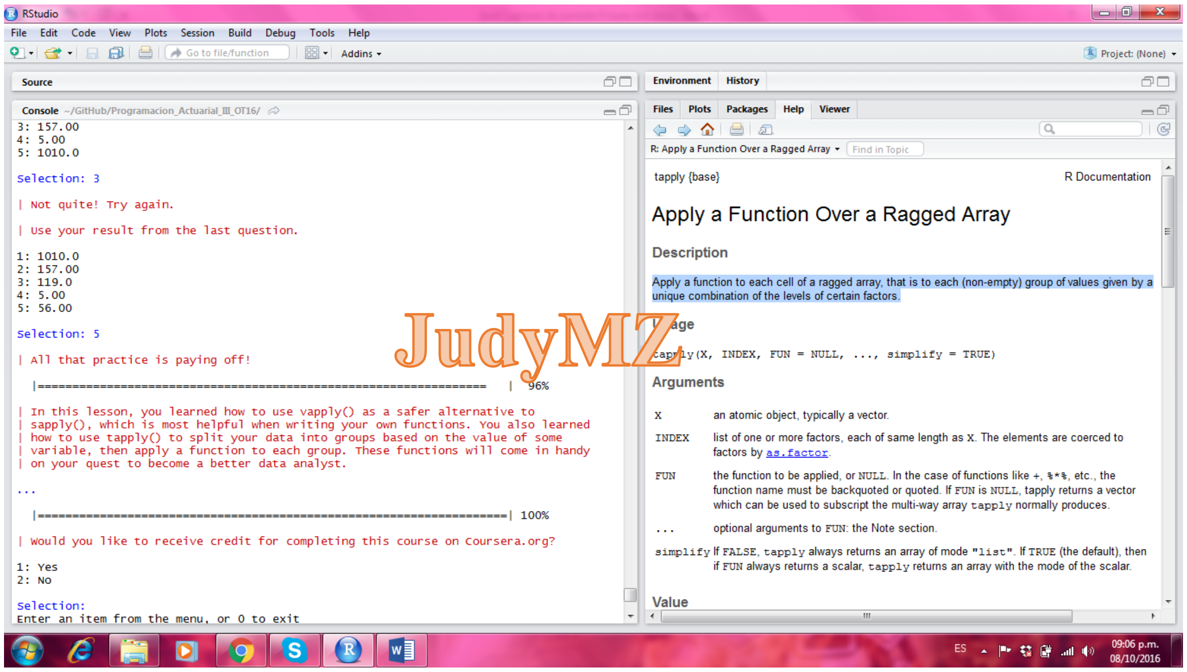
```
which(x, arr.ind = FALSE, useNames = TRUE)
arrayInd(ind, .dim, .dimnames = NULL, useNames = FALSE)
```

Arguments

| | |
|---|---|
| `x` | a `logical` vector or array. `NA`s are allowed and omitted (treated as if `FALSE`). |
| `arr.ind` | logical; should **arr**ay **ind**ices be returned when `x` is an array? |
| `ind` | integer-valued index vector, as resulting from `which(x)`. |
| `.dim` | `dim(.)` integer vector |
| `.dimnames` | optional list of character `dimnames(.)`. If `useNames` is true, to be used for constructing dimnames for `arrayInd()` (and hence, `which(*, arr.ind=TRUE)`). If `names(.dimnames)` is not empty, these are used as column names. `.dimnames[[1]]` is used as row names. |
| `useNames` | logical indicating if the value of `arrayInd()` should have (non-null) dimnames at all. |

Value

## 11: vapply and tapply



```
3: 157.00
4: 5.00
5: 1010.0

Selection: 3

| Not quite! Try again.

| Use your result from the last question.

1: 1010.0
2: 157.00
3: 119.0
4: 5.00
5: 56.00

Selection: 5

| All that practice is paying off!

  |=================================================================     |  96%

| In this lesson, you learned how to use vapply() as a safer alternative to
| sapply(), which is most helpful when writing your own functions. You also learned
| how to use tapply() to split your data into groups based on the value of some
| variable, then apply a function to each group. These functions will come in handy
| on your quest to become a better data analyst.

...

  |======================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No

Selection:
Enter an item from the menu, or 0 to exit
```

tapply {base} R Documentation

Apply a Function Over a Ragged Array

Description

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

Usage

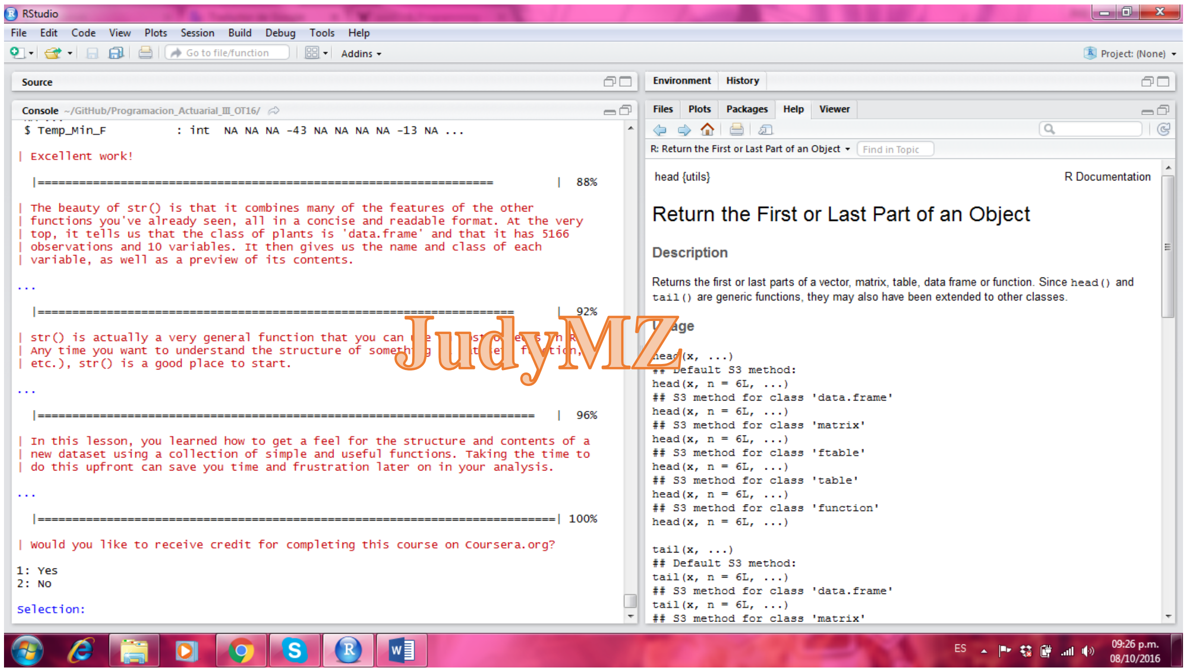
```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

Arguments

| | |
|---|---|
| X | an atomic object, typically a vector. |
| INDEX | list of one or more factors, each of same length as X. The elements are coerced to factors by as.factor. |
| FUN | the function to be applied, or NULL. In the case of functions like +, %*%, etc., the function name must be backquoted or quoted. If FUN is NULL, tapply returns a vector which can be used to subscript the multi-way array tapply normally produces. |
| ... | optional arguments to FUN: the Note section. |
| simplify | If FALSE, tapply always returns an array of mode "list". If TRUE (the default), then if FUN always returns a scalar, tapply returns an array with the mode of the scalar. |

Value

## 12: Looking at Data



```
 $ Temp_Min_F           : int  NA NA NA -43 NA NA NA NA -13 NA ...

| Excellent work!

  |=================================================================         |  88%

| The beauty of str() is that it combines many of the features of the other
| functions you've already seen, all in a concise and readable format. At the very
| top, it tells us that the class of plants is 'data.frame' and that it has 5166
| observations and 10 variables. It then gives us the name and class of each
| variable, as well as a preview of its contents.

...

  |====================================================================      |  92%

| str() is actually a very general function that you can use on most objects in R.
| Any time you want to understand the structure of something (a dataset, function,
| etc.), str() is a good place to start.

...

  |======================================================================    |  96%

| In this lesson, you learned how to get a feel for the structure and contents of a
| new dataset using a collection of simple and useful functions. Taking the time to
| do this upfront can save you time and frustration later on in your analysis.

...

  |==========================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No

Selection:
```

head {utils} R Documentation

Return the First or Last Part of an Object

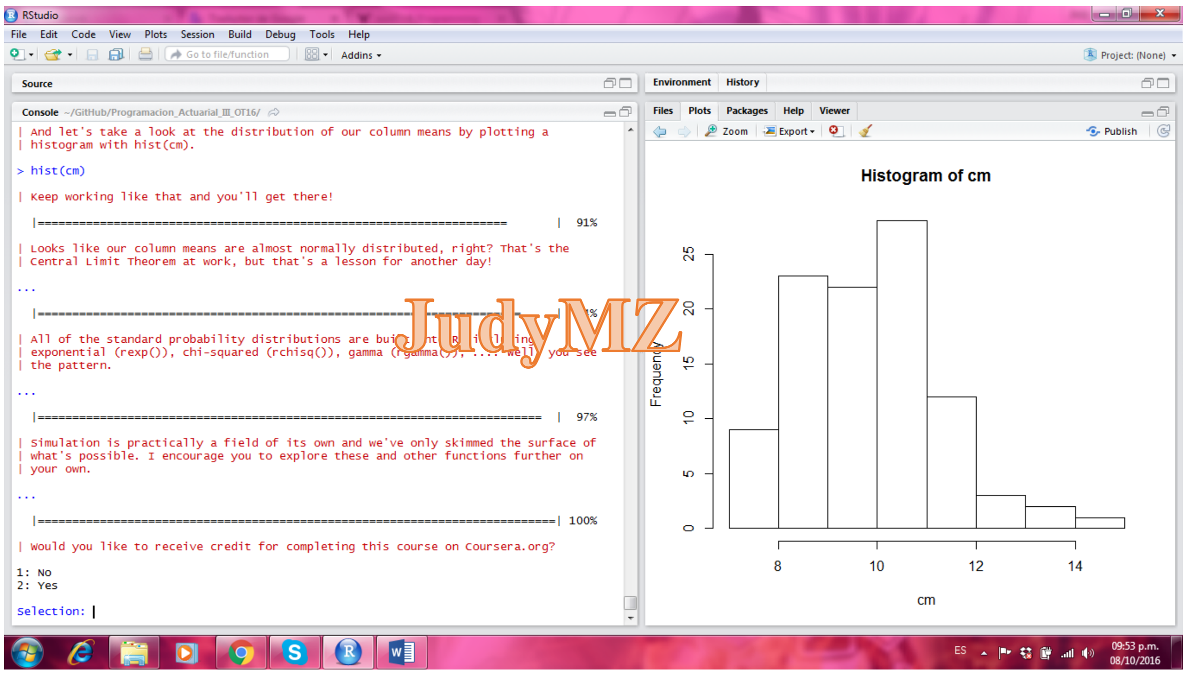
Description

Returns the first or last parts of a vector, matrix, table, data frame or function. Since head() and tail() are generic functions, they may also have been extended to other classes.

Usage

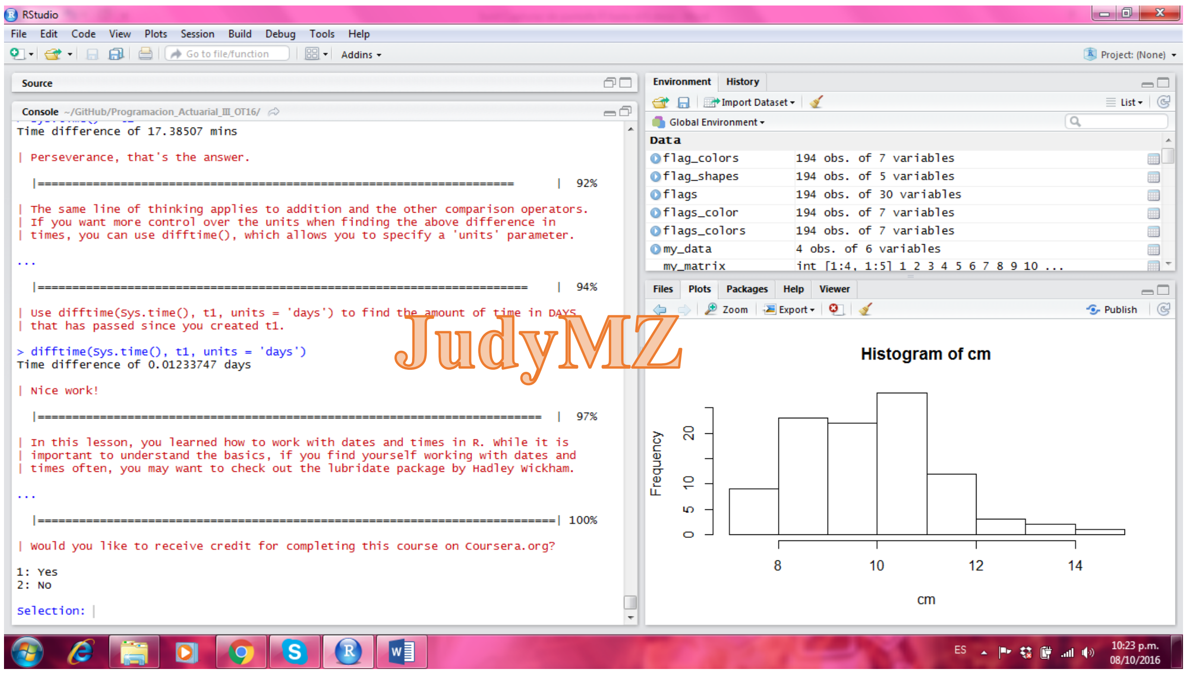
```
head(x, ...)
## Default S3 method:
head(x, n = 6L, ...)
## S3 method for class 'data.frame'
head(x, n = 6L, ...)
## S3 method for class 'matrix'
head(x, n = 6L, ...)
## S3 method for class 'ftable'
head(x, n = 6L, ...)
## S3 method for class 'table'
head(x, n = 6L, ...)
## S3 method for class 'function'
head(x, n = 6L, ...)

tail(x, ...)
## Default S3 method:
tail(x, n = 6L, ...)
## S3 method for class 'data.frame'
tail(x, n = 6L, ...)
## S3 method for class 'matrix'
```

# 13: Simulation



```
| And let's take a look at the distribution of our column means by plotting a
| histogram with hist(cm).

> hist(cm)

| Keep working like that and you'll get there!

  |==============================================================        |  91%

| Looks like our column means are almost normally distributed, right? That's the
| Central Limit Theorem at work, but that's a lesson for another day!

...

  |==================================================================    |  94%

| All of the standard probability distributions are built into R, including
| exponential (rexp()), chi-squared (rchisq()), gamma (rgamma()), .... Well, you see
| the pattern.

...

  |====================================================================  |  97%

| Simulation is practically a field of its own and we've only skimmed the surface of
| what's possible. I encourage you to explore these and other functions further on
| your own.

...

  |======================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: No
2: Yes

Selection: |
```

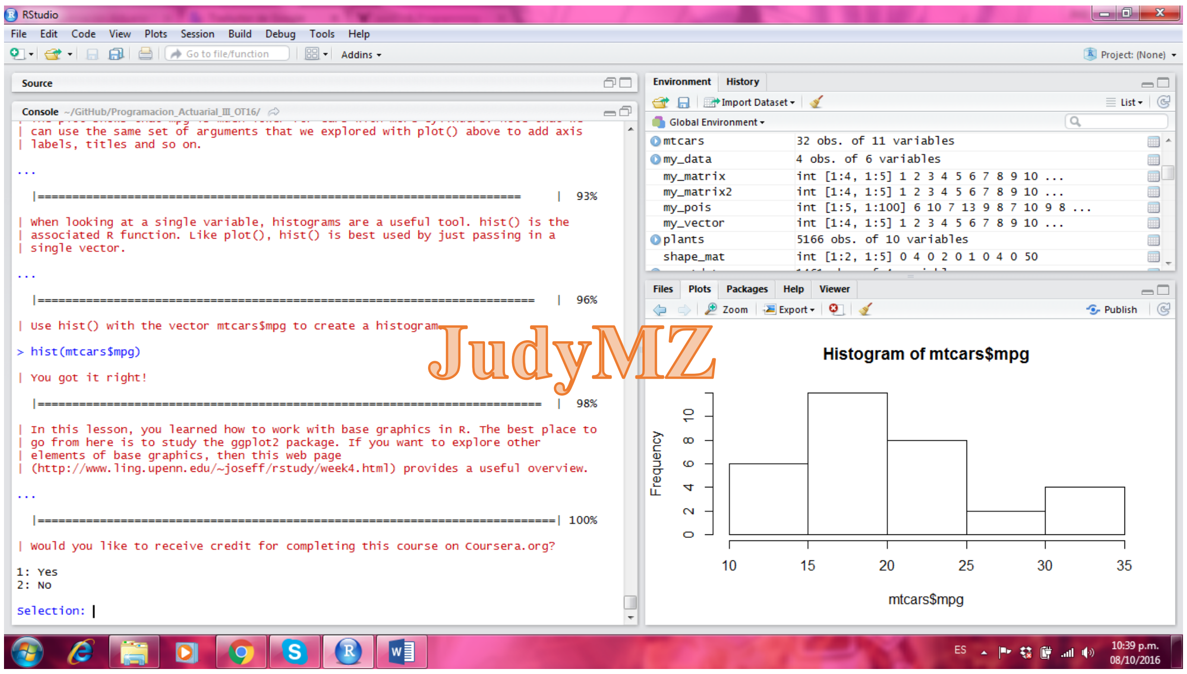
# 14: Dates and Times



```
Time difference of 17.38507 mins

| Perseverance, that's the answer.

  |===============================================================       |  92%

| The same line of thinking applies to addition and the other comparison operators.
| If you want more control over the units when finding the above difference in
| times, you can use difftime(), which allows you to specify a 'units' parameter.

...

  |=================================================================     |  94%

| Use difftime(Sys.time(), t1, units = 'days') to find the amount of time in DAYS
| that has passed since you created t1.

> difftime(Sys.time(), t1, units = 'days')
Time difference of 0.01233747 days

| Nice work!

  |====================================================================  |  97%

| In this lesson, you learned how to work with dates and times in R. While it is
| important to understand the basics, if you find yourself working with dates and
| times often, you may want to check out the lubridate package by Hadley Wickham.

...

  |======================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No

Selection: |
```

```
Data
flag_colors     194 obs. of 7 variables
flag_shapes     194 obs. of 5 variables
flags           194 obs. of 30 variables
flags_color     194 obs. of 7 variables
flags_colors    194 obs. of 7 variables
my_data         4 obs. of 6 variables
my_matrix       int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
```

# 15: Base Graphics



```
| can use the same set of arguments that we explored with plot() above to add axis
| labels, titles and so on.

...

  |=====================================================================      |  93%

| When looking at a single variable, histograms are a useful tool. hist() is the
| associated R function. Like plot(), hist() is best used by just passing in a
| single vector.

...

  |=======================================================================    |  96%

| Use hist() with the vector mtcars$mpg to create a histogram.

> hist(mtcars$mpg)

| You got it right!

  |========================================================================   |  98%

| In this lesson, you learned how to work with base graphics in R. The best place to
| go from here is to study the ggplot2 package. If you want to explore other
| elements of base graphics, then this web page
| (http://www.ling.upenn.edu/~joseff/rstudy/week4.html) provides a useful overview.

...

  |===========================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes
2: No

Selection: 
```

```
mtcars        32 obs. of 11 variables
my_data       4 obs. of 6 variables
my_matrix     int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
my_matrix2    int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
my_pois       int [1:5, 1:100] 6 10 7 13 9 8 7 10 9 8 ...
my_vector     int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
plants        5166 obs. of 10 variables
shape_mat     int [1:2, 1:5] 0 4 0 2 0 1 0 4 0 50
```

(Judy Esperanza Motero Zapata)