# Introduction to R

## Purpose of this part of the course

After this section of the course:

You should be able to

- Read data in and output data from R
- Manipulate data in R
- Use some basic R functions (e.g. basic statistics and plotting)

You *may* be able to

- Write basic computer programs in R

Particualrly if you don't have a computational background you may find that R is a 'use it or lose it' skill

There is a lot about R you will not learn in this course – much more available in books, tutorials, online forums, R demos, help pages etc.

## What is R?

Statistics package (a GNU project based on the S language)

- Statistical environment
- Graphics package
- Programming language

Completely free!

## Help( )

> help (*topic*)

or just

> *?topic*

Help even when you don’t quite know what you are looking for:

> ??*topic*

e.g.

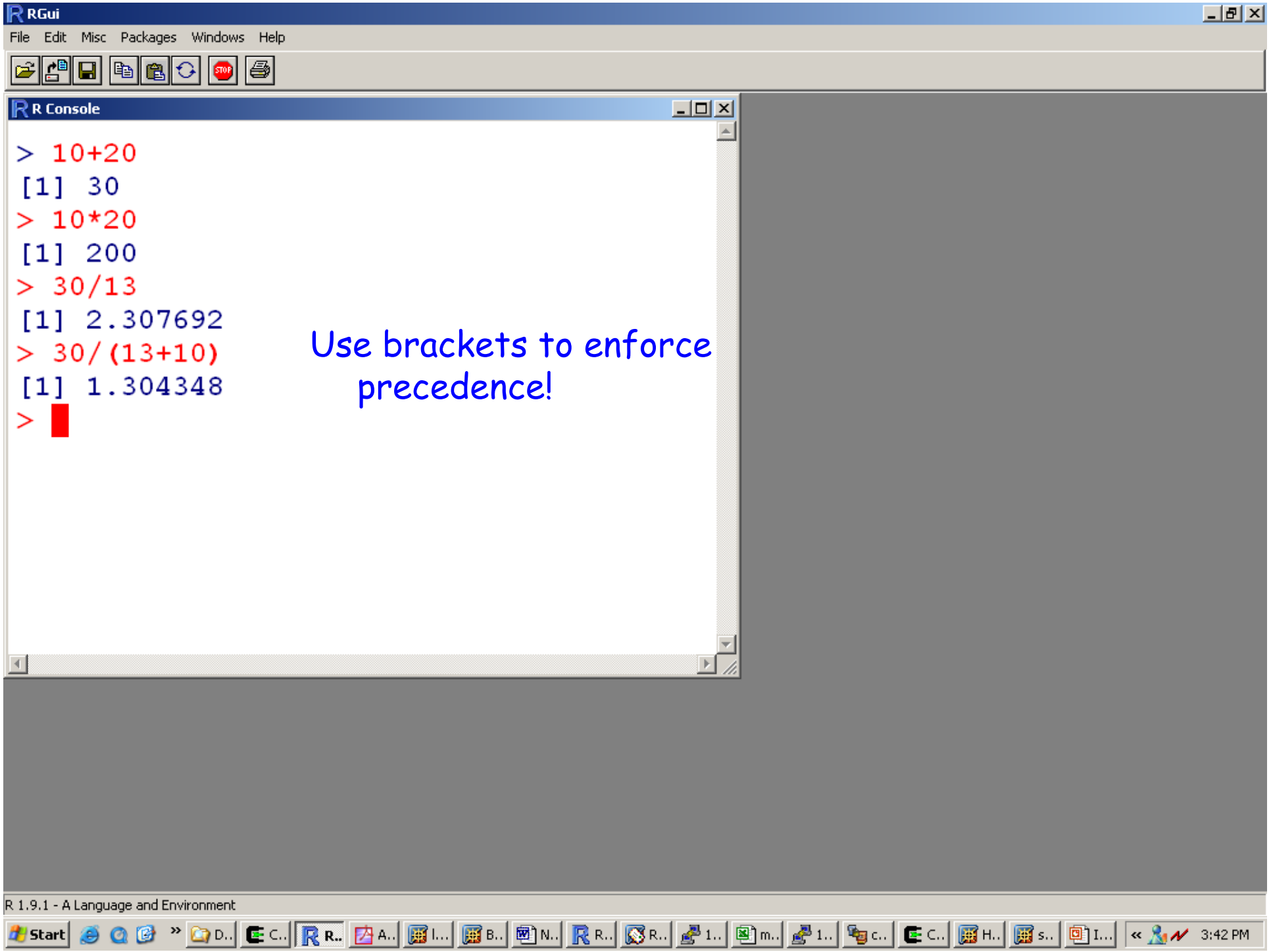
> ??”standard deviation”

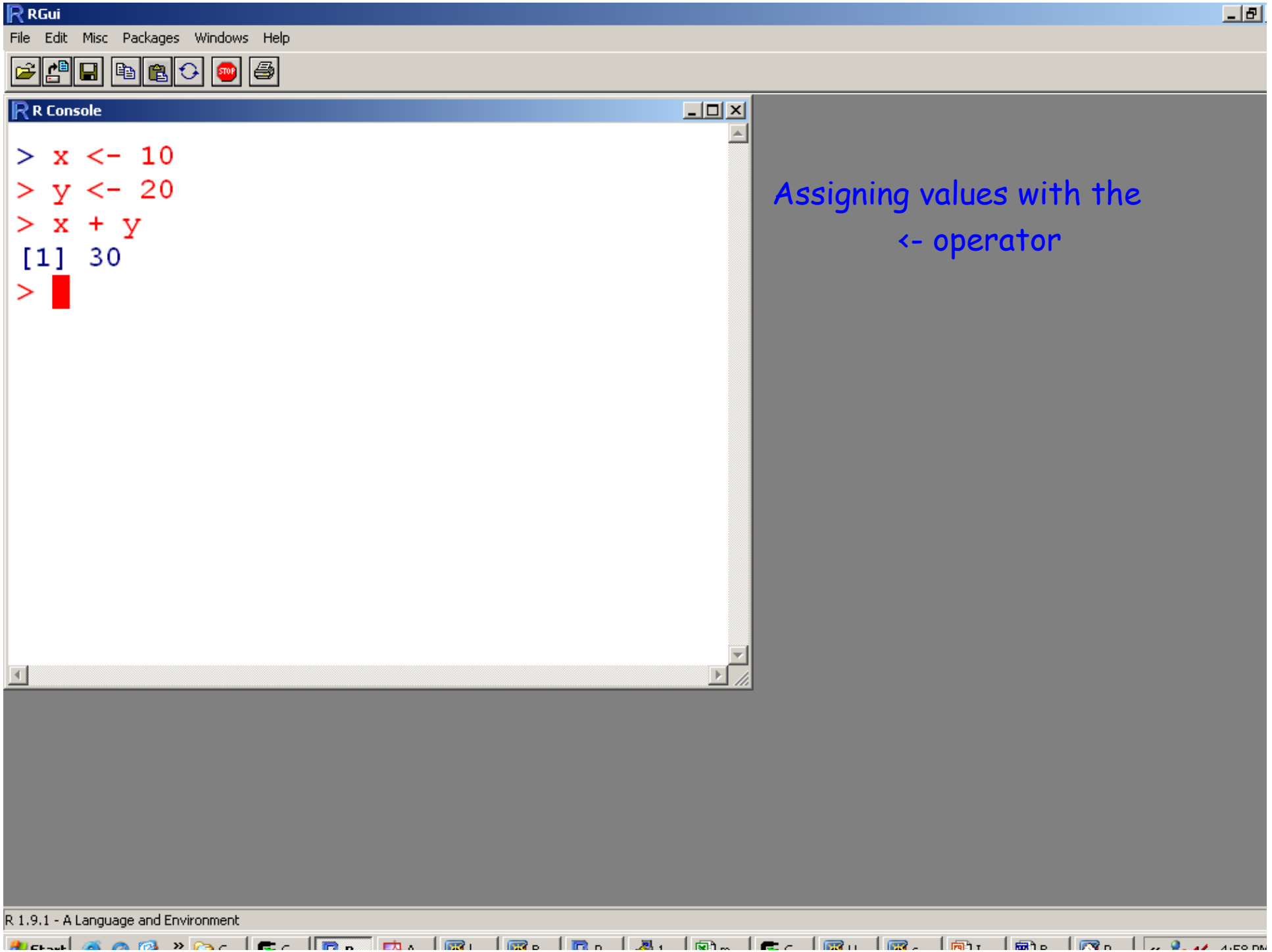
help( ) is an R function

R functions take arguments (pieces of information that you put into the function which go in between brackets and are separated by commas) and can perform a range of tasks. In the case of the ‘help’ function the task is to display information from the R documentation files.

## R as calculator

R will evaluate basic calculations which you type into the console (input window)


RGui
File Edit Misc Packages Windows Help
R Console
> 10+20
[1] 30
> 10*20
[1] 200
> 30/13
[1] 2.307692
> 30/(13+10)
[1] 1.304348
>
Use brackets to enforce precedence!
R 1.9.1 - A Language and Environment
Start
3:42 PM




Assigning values with the
<- operator

In the previous example x and y are variables. We obtained the sum of x and y by typing

x + y

In the same way we could carry out much more complicated calculations

Generally you can obtain the number (or other value) stored in any variable by typing the name of the variable, followed by enter (or by typing print (*variable*) or show (*variable*))

## Simple operations

Add: 10 + 20

Multiply: 10 * 20

Divide: 10/20

Raise to a power: 10**3.5

## Vectors

- Can think of vectors as being equivalent to a single column of numbers in a spreadsheet.

- You can create a vector using the c( ) function (*concatenate)* as follows:

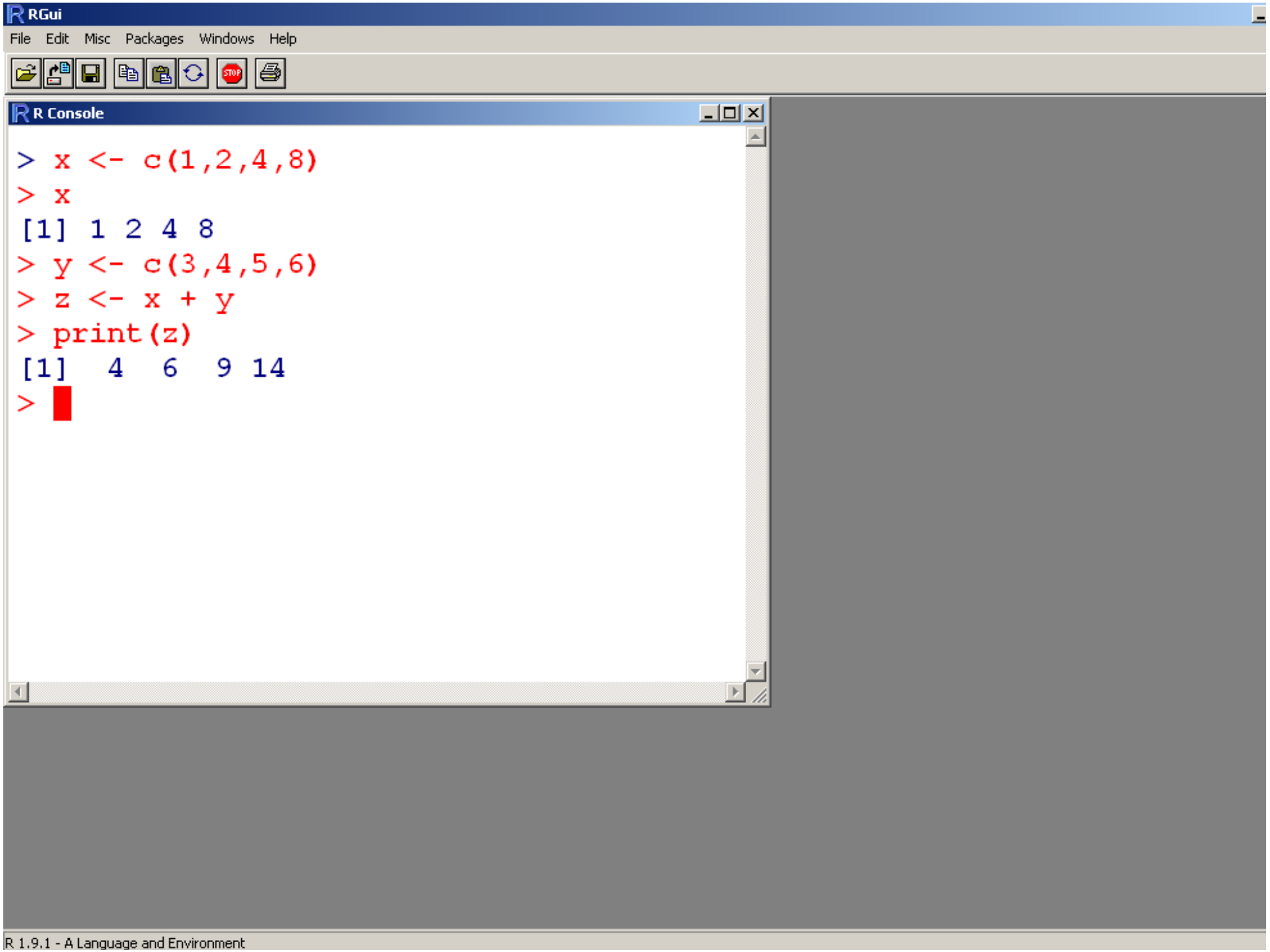
  x ← c( )

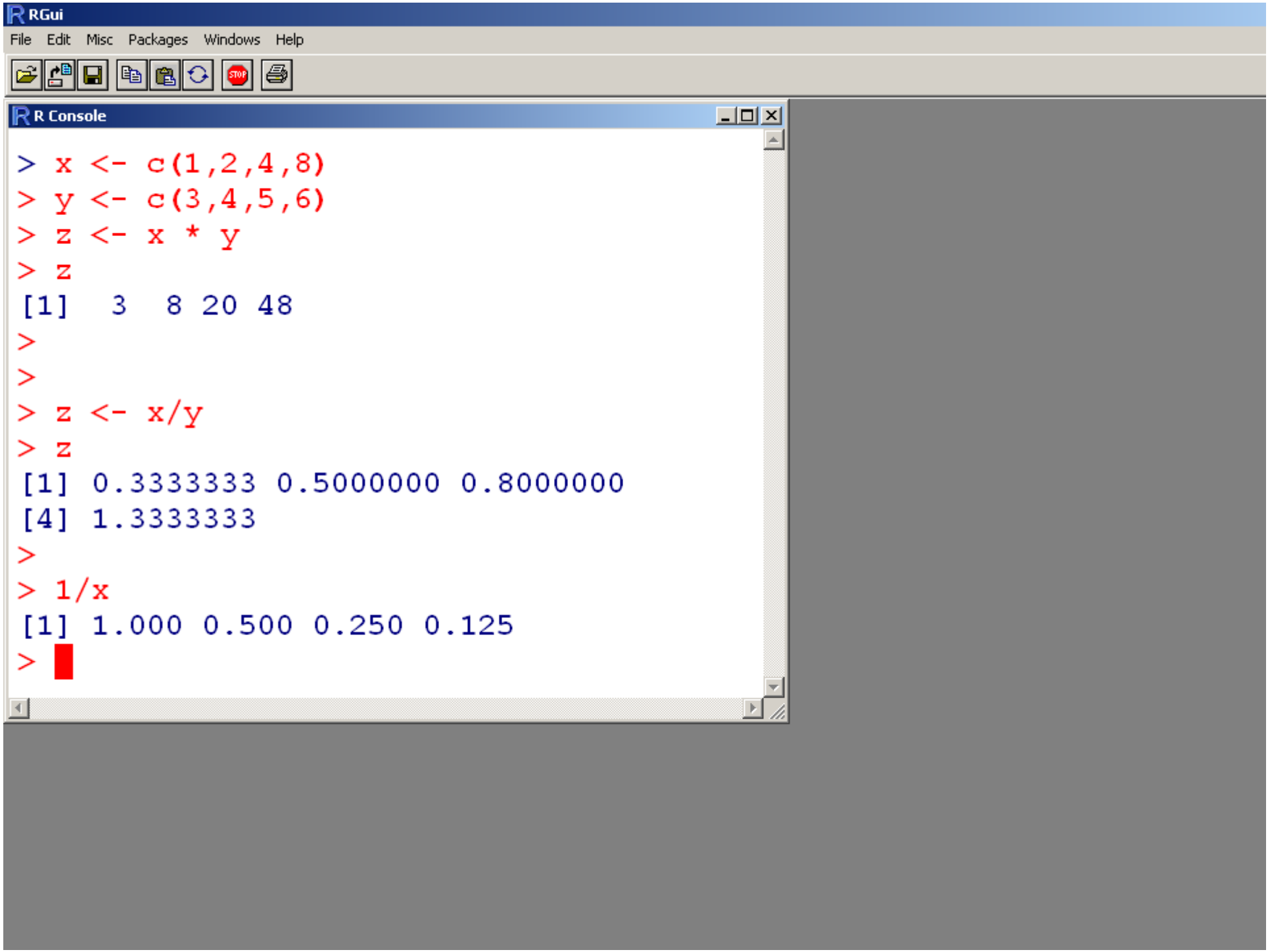
  - e.g. x ← c(1,2,4,8) creates a column of the numbers 1,2,4,8

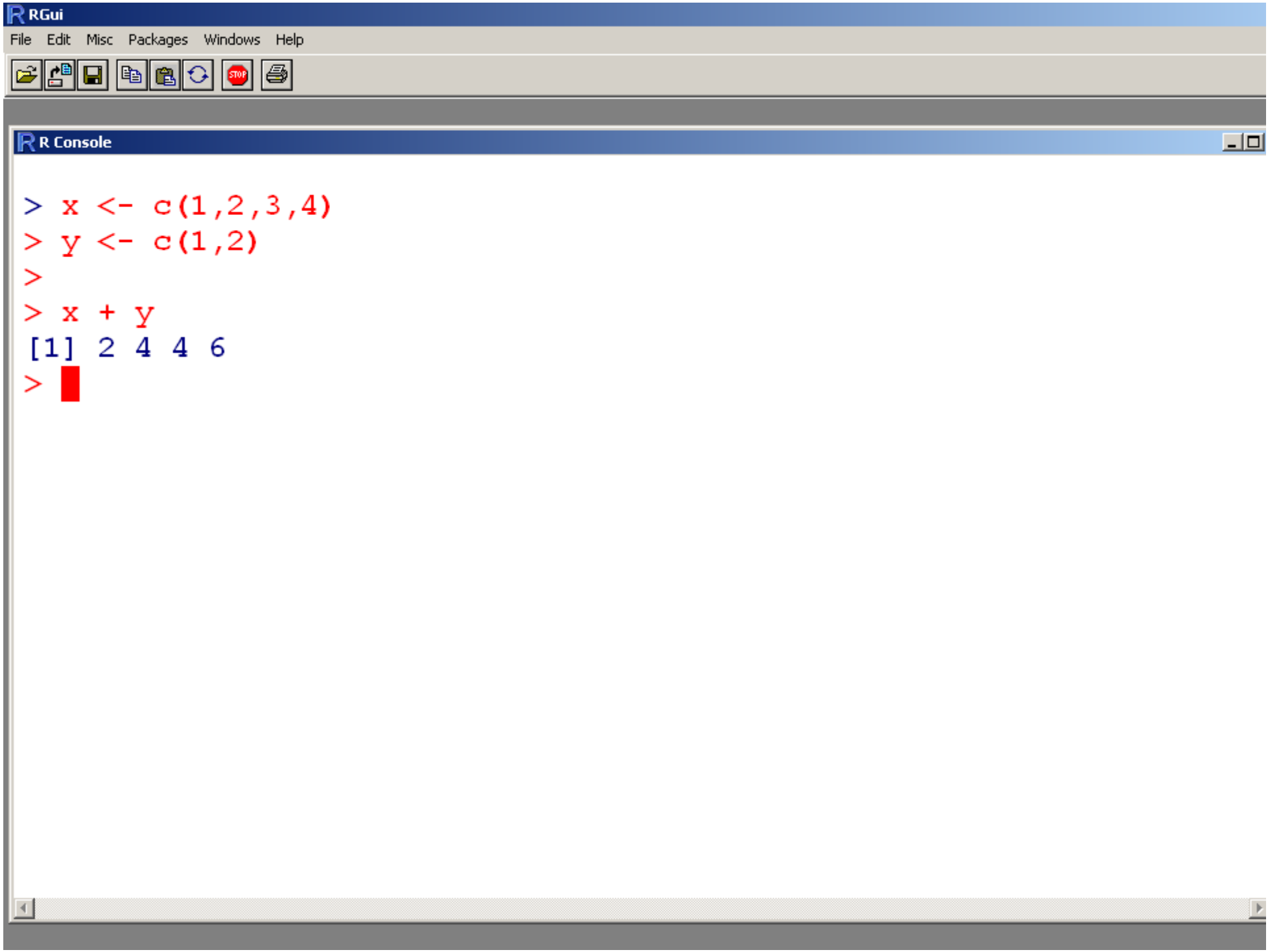
## Performing simple operations on vectors

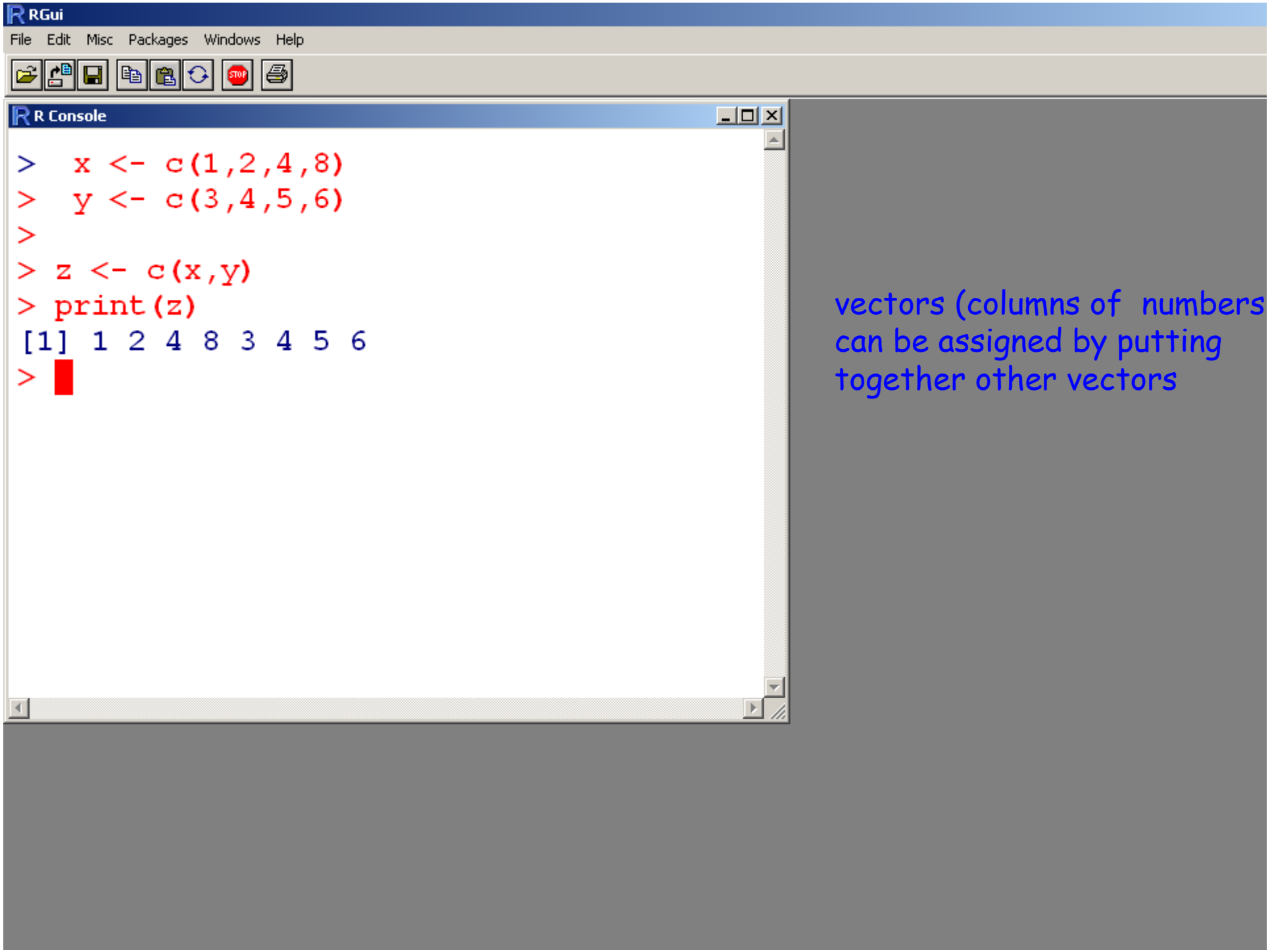
- When you carry out simple operations (+ - * /) on vectors in R that have the same number of entries R just performs the normal operations on the numbers in the vector entry by entry

- If the vectors don’t have the same number of entries then R will cycle through the vector with the smaller number of entries


RGui
File Edit Misc Packages Windows Help
R Console
> x <- c(1,2,4,8)
> x
[1] 1 2 4 8
> y <- c(3,4,5,6)
> z <- x + y
> print(z)
[1] 4 6 9 14
>
R 1.9.1 - A Language and Environment


RGui
File Edit Misc Packages Windows Help
R Console
> x <- c(1,2,4,8)
> y <- c(3,4,5,6)
> z <- x * y
> z
[1] 3 8 20 48
>
>
> z <- x/y
> z
[1] 0.3333333 0.5000000 0.8000000
[4] 1.3333333
>
> 1/x
[1] 1.000 0.500 0.250 0.125
>


RGui
File Edit Misc Packages Windows Help
R Console
> x <- c(1,2,3,4)
> y <- c(1,2)
>
> x + y
[1] 2 4 4 6
>


RGui
File Edit Misc Packages Windows Help
R Console
> x <- c(1,2,4,8)
> y <- c(3,4,5,6)
>
> z <- c(x,y)
> print(z)
[1] 1 2 4 8 3 4 5 6
>
vectors (columns of numbers
can be assigned by putting
together other vectors

## Some more useful ways of creating columns of numbers (vectors)

The seq function

seq(1,10,1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

seq(1,4,0.5) = 1, 1.5, 2, 2.5, 3, 3.5, 4

x:y

1:10 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

The rep function

rep(2,4) = 2, 2, 2, 2

Referencing specific ‘cells’ in your column of numbers (*indexing*)*:*

Example

if x is the vector 1, 2, 5 then x [1] = 1, x [2] = 2, x [3] = 5

We can even put a vector between the square brackets to refer to more than one cell of the vector

## Examples

let x be the vector 1, 2, 5 (as before) and y be the vector 1, 3

x [y] = 1, 5

x [c (y, y) ] = 1, 5, 1, 5

x [c (1, 3)] = 1, 5

x [1:2] = 1, 2

x [2:3] = 2, 5

## More useful stuff…

Instead of putting a number between the square brackets to refer to a cell(s) you can put a *condition*

### Example

let *x* be the vector 1, 0, 8, 12, 0

suppose we would like to refer to only the positive elements of *x* we can do this as

$$x\,[x > 0] = 1, 8, 12$$

You don’t have to restrict yourself to one condition/index – you can have multiple conditions (and even multiple sets of square brackets)

Example

let *x* be the vector 1, 0, 8, 12, 0

suppose we would like to refer to only the positive and even elements of *x.* we could do this as follows:

*x* [*x* > 0 & *x* %% 2 = = 0] = 8, 12

Notes:

1. in R, as in many computer languages == compares, while a single = assigns!

2. %% is the modulus operator in R

## Example

let *x* be the vector 1, 0, 8, 12, 0

suppose we would like to refer to only the positive elements among the first three elements of *x* we can do this with this somewhat convoluted expression

*x* [1:3][*x*[1:3] > 0] = 1, 8

*(not recommended unless you know what you are doing)*

Note: vectors can consist of characters (i.e. letters/words) instead of numbers

e.g. *x* <- c (“purple” , “green”, “blue”)

BUT note: as far as R is concerned a vector never consists of numbers AND characters. If you have numbers and characters in your vector, R will consider every entry in your vector to be a character(s) (i.e. it will stop thinking of the numbers as numbers and instead just think of them as words – this will, of course mean that you can not add them, for example)

# Some more types of data (I)

## Lists

A set of objects (e.g. vectors) can be combined under a single name as a list (somewhat similar to a spreadsheet in Excel)

*Example*

x ← c (1, 7, 8, 9, 10)

y ← c (“red”, “yellow”, “blue”, “green”)

example_list ← list (size = x, colour = y)

## Some more types of data (II)

### Data frames (and the function *data.frame ( )* )

A special kind of list in which the entries in a specific position in the elements of the list correspond to one another. Each element of the list has the same length

### Example

```
h <- c (150, 170, 168, 179, 130)

w <- c (65, 70, 72, 80, 51)

patient_data <- data.frame (weights=w, heights=h)
```

Accessing a particular column, cell or row of a data frame

*Accessing a cell*

patient_data [ *i*, *j* ]

*($j^{th}$ cell in the $i^{th}$ column)*

*Accessing a column*

patient_data [ , *i* ]

*(column i)*

*Accessing a row*

patient_data [ *i*,  ]

(*row i)*

## Some more types of data (III)

### Factors

A factor is a vector that usually encodes information about the group to which a particular observation belongs.

e.g. suppose you have measure heights of Irish and British people and record the results as follows

```
heights <- c(1.7,1.95,1.63,1.54,1.29)
```

You make a new vector fac_heights to record the nationality that each observation pertains to

```
fac_heights <- factor(c("GB", "IR", "GB", "GB", "IR"))
```

This kind of data object will be very useful when we try to do some statistical tests for differences between groups

## Finding out about a data object

mode ( ): tells you the storage ‘mode’ of an object (i.e. whether it is a numeric vector, or a list etc.)

class( ): provides information about the object’s class. The class of an object often determines how the data object is handled by a function.

You can also set the object’s mode, attributes or class using the above functions.

e.g. mode (x) <- “numeric”

# Data input and output

## Topics

- Data that comes with R
- Inputting data from a file
- Writing data to a file
- Writing data to the clipboard
- NB: saving the workspace

R comes with several pre-packaged datasets

You can access these datasets with the *data* function

data ( ) *gets you a list of all the datasets*

data (Titanic) *loads a dataset about passengers on the Titanic (for example)*

summary (Titanic) *provides some summary information about the dataset Titanic*

attributes(Titanic) *provides some more information*

*Typing the dataset name on its own (followed by* Enter) *will display the data*

## Getting data from a file (the *file.choose* and *scan* functions)

The function file.choose ( ) allows you to browse and select a file

The function scan ( ) allows you to read in data from a file

The command

```
q <- scan ( file.choose ( ) )
```

will allow you to browse to a file and will then scan in all the entries in the file (space or newline separated) to the vector q

```
q <- scan ( file.choose ( ) , sep = ".")
```

will scan entries separating by the "." character (or newline)

Note: "\t"  is the symbol for the 'Tab' character

## Scanning a list from a file

```
q <- scan ( file.choose ( ) , what=list(names="n", ages=0), sep = "\t")
```

## Getting a data frame from a file

Can use the function read.table ( ) to read in a dataframe:

```
q <- read.table ( file.choose ( ) , sep = “\t”, header = T)
```

OR use scan ( )

```
q <- scan ( file.choose ( ) , what=data.frame(names=“n”, ages=0), sep
= “\t”)
```

## Writing data to a file (the *write* and *write.table* functions)

Change directory on the file menu then

write ( q, file = “filename”, ncol = 2)

*(for vector, ncol specifies the number of columns in output)*

write.table (q, file = “filename” )

*(works quite well for a data frame)*

*as always there are many optional arguments*

Save your workspace!

save.image ( “filename”)

(conventionally, using file extension .r or .rdata)

# Writing your own functions in R

## Loops (*for*)

Syntax:

let *x* be a vector

for ( *i* in *x* )

command

e.g. try for(*i* in 1:10)

print(i)

## Loops

Example: add the elements of a vector x ← c (1, 2, 8)

```
sum ← 0
for (i in x)
   sum ← sum + i
```

sum = 11

Note: of course there is a function, sum(), that does this for you

## Conditional execution (*if*)

Syntax:

if ( *condition* )

*command*

## Conditional execution

Example

```
x ← 10
if ( x > 2 )
  print ( x )
```

## Conditional execution

>, <, <=, >=, !=, ==  are all used to compare numbers (NB note that to compare whether two numbers are the same you must use the double equals to symbol)

These symbols can also be used in conditional indexing (see earlier)

You can combine conditions with the & or | sign

## Loop & conditional execution

Example: add the positive elements of a vector x ← c (1, -2, 8)

```
pos_sum ← 0
for (i in x)
  if (i > 0)
  pos_sum ← pos_sum + ì
```

pos_sum = 9

*This is for illustration only – there are quicker ways of doing this in R*

## Blocks

Sometimes you need to execute several commands in a loop or after testing a condition

R will group together all commands within { }

Add up all the numbers between 1 and 10 and multiply by the sum of the numbers between 5 and 10

```
for ( i in 1:10) {
m <- m + i
      if ( i > 5 ) {
      k <- k + i
      }
}
s <- m * k
print (s)
```

*Notice the indentation – this makes it easier to read an R script*

Making your own function:

```
my_function <- function ( … ) {}
fix ( my_function )
```

Note: it’s important to comment your code (anything after the # symbol is ignored by R – this is a place to put ‘notes to self’)

Example: write a function to calculate the average of the positive numbers in a vector *x*

```
pos_average ← function ( x )  {

pos_sum ← 0

        for ( i in x ) {

                if ( i > 0 ) {

                pos_sum ← pos_sum + i       # store sum of positives

                pos_count ← pos_count + 1  # incremment counter

                }

        }

return(pos_sum/pos_count)

}
```

*Again, this is for illustration only – there are quicker ways of doing this in R*

# R for graphics

A scatter plot between two variables

plot(x,y)

(*many options*)

Histogram

hist(x)

(*many options*)

Demonstrate R’s graphics capabilities

demo(graphics)