Lesson 1: A first R session

Objectives

- Familiarize ourselves with the R console
  - prompts
  - comments
  - entering commands
  - storing values
  - using the history
  - tab-completion
  - getting help
  - quitting

Getting Started

To begin with, we're going to run R in a terminal. You will usually be using a more sophisticated and user-friendly interface, but for this first session we'll use the bare-bones version.

From the terminal, R is started by entering the letter R at the prompt. R responds with:

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i486-pc-linux-gnu (32-bit)

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.

>
```

You'll see this everytime you start R, so most of us get in the habit of ignoring it. However, this header contains some important information:

- which version of R you are running. If it's more than six months old, chances are a newer version is available. It's usually not a problem to run a *slightly* older version. But if you want to use actively developed packages, you'll save some head-aches by staying up to date.
- get citation information with the function `citation()`
- view some demonstrations with `demo()`
- get help with `help()` or `help.start()` (more on this below)

Talking to R

After all the boilerplate, you'll see your cursor sitting beside the prompt:

```
>
```

The > at the beginning of a line is how R indicates it's ready to recieve instructions. We can start with some basic math:

```
2 + 3
```

```
[1] 5
```

```
4 * 5
```

```
[1] 20
```

```
pi
```

```
[1] 3.1416
```

That's not a lot of accuracy for `pi`! Not to worry, behind the scenes, R uses very high precision values for all calculations. You can set the number of digits to display on the screen with the `options(digits=<n>)` function:

```
options(digits = 16)
pi
```

```
[1] 3.141592653589793
```

```
options(digits = 5)
pi
```

```
[1] 3.1416
```

```
cos(pi)
```

```
[1] -1
```

```
sin(pi)
```

```
[1] 1.2246e-16
```

If you remember trigonometry, this last result may worry you. `sin(pi)` should be 0. R thinks it's actually a super-tiny number that's not quite zero. This is a consequence of how computers deal with decimal values. Most consumer software will hide this from you, but not R. There are ways to cope with this in the rare instances where it matters, but we'll leave that for now.

Next, try the following:

```
tan(pi
```

Notice how the prompt has now changed:

```
+
```

The + symbol at the beginning of the line indicates that R is waiting for you to complete the command. Once you add the closing) you'll get the answer.

[1] Indices

You'll have noticed by now that each line of output from R starts with `[1]`. This is actually a helpful visual index to help you scan the results. Try this:

```
1:10
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

The `:` operator generates a sequence of numbers. Now try:

```
1:100
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
 [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
 [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
 [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
 [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

Now we can see the value of the indices. When you have output that wraps over several lines, the index at the beginning of each line tells you how far along in the output you are. In this case, it's not surprising that the 86th element is 86, but normally we won't be looking at orderly sequences of numbers.

This also brings up another aspect of R. There are no single numbers (scalars) in R. All numbers are part of a vector, although vectors may have only a single number in them. This has interesting properties which we'll investigate later.

Comments

When you're learning R, you'll spend a lot of time learning how to communicate with the program. But you will also want to leave notes for yourself in the code tha you write. To do this, R uses the # symbol to signal a line of text that the program should ignore. Throughout this workshop you'll see lines in our examples that start with a #. These are messages to you, the reader, that the computer will happily ignore.

```
## This is how R does exponents:
2^3
```

```
[1] 8
```

```
## And this is how it does logarithms:
log(10)
```

```
[1] 2.3026
```

```
## NB: log() means ln in R!
log10(10)  ## Here's log base 10!
```

```
[1] 1
```

Notice that everything after the # is ignored - you can even put comments on the right side of a line that has code on the left.

Variables

You can store results temporarily by assigning the output of a computation to a *variable*.

```
## use the = sign to store the value of the calculation in the variable 'a'
a = exp(1)

## exp(n) is a function to return the value of e raised to the power n

## you can use '<-' instead of '='. They do the same thing in this context.
b <- 42
c <- 2^3
```

We can now recall the values of these previous calculations by using the variables:

```
a
```

```
[1] 2.7183
```

```
b
```

```
[1] 42
```

```
c
```

```
[1] 8
```

We can now combine these variables into more complicated calculations:

```
## We can combine our variables into complex equations:
(-b + sqrt(b^2 - 4 * a * c))/(2 * a)  ## the quadratic equation
```

```
[1] -0.19288
```

History and Tab-Completion

To complete the solution, we need to repeat the calculation for the minus side. This is a tedious long calculation to type in. R provides an easy way to modify and repeat previous commands. At the prompt, press the up arrow. This will add the previous command to the prompt. Pressing the up arrow again cycles you backwards through the history of commands you've entered. The down arrow cycles you forward. So once the previous equation is on the line, switch the code from `+ sqrt` to `- sqrt` and you've got the other half of the solution.

You can even assign the result to a new variable:

```
quadratic_equation.solution = (-b + sqrt(b^2 - 4 * a * c))/(2 * a)
```

You can use upper- and lower-case letters, underscores '_', and periods in your variable names. You can also use numbers, but not at the start of the name.

This is a long name to type out repeatedly. R can help you with that. At the prompt, type q, then hit the tab key twice. R shows you all the different objects it knows that start with q. Now continue typing until you have quadr, then hit tab again. R only knows one object that starts with these five letters, so it auto-completes it for you.

Workspace Management

All of the variables you've created are temporarily stored in your workspace (technically, the global environment).

You can list all of the objects in your current environment with ls():

```
ls()
```

To remove object use rm():

```
x <- 5
rm(x)
x
```

```
Error: object 'x' not found
```

x has been deleted. You can remove all objects from your current environment at once with:

```
rm(list = ls())
```

Help

Before we finish our first session, try out the built in help:

```
help.start()
```

This opens a window in your web browser with links to all the R documentation. Note the address: 127.0.0.1. This is your computer, all of these files are local, you don't need an internet connection to get to them.

You can also search for help directly from R:

```
help(t.test)
```

This opens the help for t.test using the program less. You can return to R by hitting q.

A shortcut for 'help(t.test) is:

```
?t.test
```

If you don't know the name of the help topic you need, you can use this:

```
help.search("phylogeny")  ## note the quotations
## or another shortcut:
??phylogeny
```

Quitting

Finally, when you're ready to quit, use this:

```
q()
```

R will ask you if you want to save your workspace. This will save all the variables you have created, and reload them next time you start R in the same directory. I strongly recommend you *DO NOT* do this. It's convenient in the short-term, but can cause head-aches when you have to manage data in complex. There are much better strategies for saving your work. We'll talk about some of them in another lesson.