CHAPTER 1

Try R

In this first chapter, we'll cover basic R expressions. We'll start simple, with numbers, strings, and true/false values. Then we'll show you how to store those values in variables, and how to pass them to functions. We'll show you how to get help on functions when you're stuck. Finally we'll load an R script in from a file.

Let's get started!

Expressions 1.1

Type anything at the prompt, and R will evaluate it and print the answer.

Let's try some simple math. Type the below command.

[Or, if you prefer, click on the command and it will be typed into the console for you!]

```
> 1 + 1
[1] 2
```

There's your result, `2`. It's printed on the console right after your entry.

Type the string `"Arr, matey!"`. (Don't forget the quotes!)

```
> "Arr, matey!"
[1] "Arr, matey!"
```

Now try multiplying 6 times 7 (`*` is the multiplication operator).

```
> 6 * 7
[1] 42
```

Logical Values 1.2

Some expressions return a "logical value". `TRUE` or `FALSE`. (Many programming languages refer to these as "boolean" values.) Let's try typing an expression that gives us a logical value:

```
> 3 < 4
[1] TRUE
```

And another logical value (note that you need a double-equals sign to check whether two values are equal - a single-equals sign won't work):

```
> 2 + 2 == 5
[1] FALSE
```

`T` and `F` are shorthand for `TRUE` and `FALSE`. Try this:

```
> T == TRUE
[1] TRUE
```

Variables 1.3

As in other programming languages, you can store values into a variable to access it later. Type `x <- 42` to store a value in `x`.

```
> x <- 42
```

`x` can now be used in expressions in place of the original result. Try dividing `x` by `2` (`/` is the division operator).

```
> x / 2
[1] 21
```

You can re-assign any value to a variable at any time. Try assigning `"Arr, matey!"` to `x`.

```
> x <- "Arr, matey!"
```

You can print the value of a variable at any time just by typing its name in the console. Try printing the current value of `x`.

```
> x
[1] "Arr, matey!"
```

Now try assigning the `TRUE` logical value to `x`.

```
> x <- TRUE
```

Functions 1.4

You call a function by typing its name, followed by one or more arguments to that function in parenthesis. Let's try using the `sum` function, to add up a few numbers. Enter:

```
> sum(1, 3, 5)
[1] 9
```

Some arguments have names. For example, to repeat a value 3 times, you would call the `rep` function and provide its `times` argument:

```
> rep("Yo ho!", times = 3)
[1] "Yo ho!" "Yo ho!" "Yo ho!"
```

Try calling the `sqrt` function to get the square root of `16`.

```
> sqrt(16)
[1] 4
```

Help 1.5

`help(functionname)` brings up help for the given function. Try displaying help for the `sum` function:

```
> help(sum)
sum                     package:base                     R Documentation

Sum of Vector Elements

Description:

     'sum' returns the sum of all the values present in its arguments.

Usage:

     sum(..., na.rm = FALSE)
...
```

(Don't worry about that optional `na.rm` argument, we'll cover that later.)

`example(functionname)` brings up examples of usage for the given function. Try displaying examples for the `min` function:

```
> example(min)

min> require(stats); require(graphics)

min>  min(5:1, pi) #-> one number
[1] 1

min> pmin(5:1, pi) #->  5  numbers
[1] 3.141593 3.141593 3.000000 2.000000 1.000000

...
```

Now try bringing up help for the `rep` function:

```
> help(rep)
rep                     package:base                     R Documentation

Replicate Elements of Vectors and Lists

Description:

     'rep' replicates the values in 'x'.  It is a generic function, and
     the (internal) default method is described here.
...
```

Files 1.6

Typing commands each time you need them only works for short scripts, of course. R commands can also be written in plain text files (with a ".R" extension, by convention) for executing later. You can run them directly from the command line, or from within a running R instance.

We've stored a couple sample scripts for you. You can list the files in the current directory from within R, by calling the `list.files` function. Try it now:

```
> list.files()
[1] "bottle1.R" "bottle2.R"
```

To run a script, pass a string with its name to the source function. Try running the `"bottle1.R"` script:

```
> source("bottle1.R")
[1] "This be a message in a bottle1.R!"
```

Now try running `"bottle2.R"`:

```
> source("bottle2.R")
[1] "Will ye be me pen pal?"
```

Chapter 1 Completed

You've reached the end of Chapter 1... where you discover a badge!

Excellent work! Now you know the basics of R expressions. You've learned how to create and access variables, and how to call functions. You've learned how to run pre-made scripts. And you've learned how to access R's help functionality when you need it.

Now it's time to learn about the features that make R really unique and useful - its data structures. Vectors are first; we'll talk about them in the next chapter!

Continue