Basic Computing 2 – Packages, Functions, Documenting code

Stefano Allesina

Basic Computing 2

- **Goal:** Show how to install, load and use the many freely available packages. Illustrate how to write user-defined functions and how to organize code. Showcase basic plotting functions. Introduce the package `knitr` for writing beautiful reports.
- **Audience:** Biologists with basic knowledge of `R`.
- **Installation:** To produce well-documented code, we need to instal the package `knitr`.

Packages

`R` is the most popular statistical computing language among biologists due to its highly specialized packages, often written from biologists for biologists. You can contribute a package, too! The `RStudio` support `goo.gl/harVqF` provides guidance on how to start developing `R` packages and Hadley Wickham's free online book `r-pkgs.had.co.nz` will make you a pro.

You can find highly specialized packages to address your research questions. Here are some suggestions for finding an appropriate package. The Comprehensive R Archive Network (CRAN) offers several ways to find specific packages for your task. You can either browse packages `goo.gl/7oVyKC` and their short description or select a scientific field of interest `goo.gl/0WdIcu` to browse through a compilation of packages related to this discipline.

From within your `R` terminal or `RStudio` you can also call the function `RSiteSearch("KEYWORD")`, which submits a search query to the website `search.r-project.org`. The website `rseek.org` casts an even wider net, as it not only includes package names and their documentation but also blogs and mailing lists related to `R`. If your research interests relate to high-throughput genomic data, you should have a look the packages provided by Bioconductor `goo.gl/7dwQlq`.

Installing a package

To install a package type

```r
install.packages("name_of_package")
```

from the console, or choose the panel **Packages** and then click on *Install* in `RStudio`.

Loading a package

To load a package type

```r
library(name_of_package)
```

or call the command into your script. If you want your script to automatically install a package in case it's missing, use this boilerplate:

```
if (!require(needed_package, character.only = TRUE, quietly = TRUE)) {
    install.packages(needed_package)
    library(needed_package, character.only = TRUE)
}
```

Example

For example, say we want to access the dataset `bacteria`, which reports the incidence of *H. influenzae* in Australian children. The dataset is contained in the package `MASS`.

First, we need to load the package:

```
library(MASS)
```

Now we can load the data:

```
data(bacteria)
bacteria[1:3,]
```

```
##   y ap hilo week  ID     trt
## 1 y  p   hi    0 X01 placebo
## 2 y  p   hi    2 X01 placebo
## 3 y  p   hi    4 X01 placebo
```

Random numbers

For statistical tests and simulations, we often need to draw random numbers. `R` has functions to sample from very many different statistical distributions. For example:

```
runif(5) # sample 5 numbers from the uniform distribution between 0 and 1
```

```
## [1] 0.7238014 0.6691725 0.6766714 0.2304258 0.8726901
```

```
runif(5, min = 1, max = 9) # set the limits of the uniform distribution
```

```
## [1] 2.437500 7.243432 8.900818 6.400428 8.386561
```

```
rnorm(3) # three values from standard normal
```

```
## [1]  1.7999900  0.1029395 -0.2343888
```

```
rnorm(3, mean = 5, sd = 4) # specify mean and standard deviation
```

```
## [1]  3.182072 10.599117  6.669041
```

To sample from a set of values, use `sample`:

```r
v <- c("a", "b", "c", "d")
sample(v, 2) # without replacement
```

```
## [1] "b" "d"
```

```r
sample(v, 6, replace = TRUE) # with replacement
```

```
## [1] "c" "a" "d" "a" "b" "b"
```

```r
sample(v) # simply shuffle the elements
```

```
## [1] "c" "a" "d" "b"
```

Writing functions

The `R` community provides about 7,000 packages. Still, sometimes there isn't an already made function capable to do what you need. In these cases, you can write your own functions. In fact, it is in general a good idea to always divide your analysis into functions, and then write a small "master" program that calls the functions and performs the analysis. In this way, the code will be much more legible, and you will be able to recycle the functions for your other projects.

To explore the use of functions, we are going to draw the population dynamics

Organizing and running code

Documenting the code using `knitr`

> *Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.*
>
> Donald E. Knuth, Literate Programming, 1984

When doing experiments, we typically keep track of everything we do in a laboratory notebook, so that when writing the manuscript, or responding to queries, we can go back to our documentation to find exactly what we did, how we did it, and possibly why we did it. The same should be true for computational work.

Exercises