R seminars series

Session 3: Introduction to basic programming in R

Laura Azzani – laura.azzani@pg.canterbury.ac.nz

Camille Coux – camille.coux@pg.canterbury.ac.nz

School of Biological Sciences,

University of Canterbury

Concepts and overview (Laura)

- Vectorisation in R
  - Fast and concise code
  - Potential danger: recycling of vectors!
- For loops
  - The classical form of iteration
  - How to make them fast
  - How to measure speed of operations in R
- Apply() family of functions
  - R's way of doing iteration
- Exercise 1

Concepts and overview (Camille)

- Control flow
  - Conditional execution (if... else...)
  - Exercise 2
- Debugging “for” loops
- Functions
  - Built-in functions
  - Help menu
  - Writing your own functions
  - Exercise 3

Vectorisation in R

- “Scalars” do not exist in R:

  everything is a vector (unless it is something bigger)

  ```
  > is.vector(5)
  [1] TRUE
  ```

- What do we mean by vector?

  A collection of elements.

- Vectors are central to R, so it shouldn't surprise us that R's functions and operators are **vectorised**

Vectorisation in R

- “Scalars” do not exist in R:

  everything is a vector (unless it is something bigger)

  ```
  > is.vector(5)
  [1] TRUE
  ```

- What do we mean by vector?

  A collection of elements.

- Vectors are central to R, so it shouldn't surprise us that R's functions and operators are **vectorised** → (= they do the same operation on a vector of values as they would do on each single value)

Vectorization

- Vectorization = concise and fast code (in R)
- Open the R script (section 1). We'll see some examples of vectorised functions and operators, and...

  One thing to look out for: Recycling!
- Recycling means literally this: re-using the SHORTER vector.

For loops

- For loops are iterations of the same action or operation over each element of a vector.

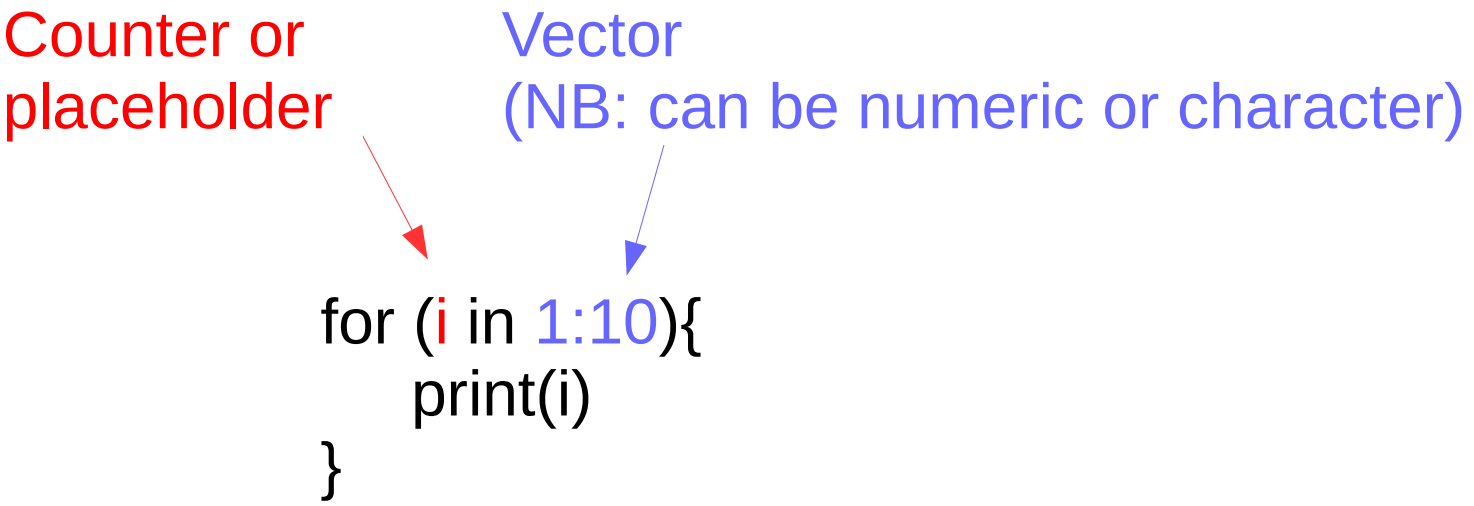
  Syntax:

  for (counter in vector) command

  or, if command takes more than one line

```
for (counter in vector){
     command1
     command2
}
```

For loops



- For each iteration (= “repetition” of the loop), the counter will assume a new value, until the end of the vector.
- let's see them in practice (Section 2, for loops in the R Script)

For loops

- Can also be used to “store” values, not just print
- In this case, you must create a “container” before calling the loop (R doesn't like to operate on “nothing”)
- Importance of memory pre-allocation: whether you start with an empty or full container, makes all the difference in terms of speed!
- We can measure how long it takes to perfom one or more operations in R using system.time()

  (open R script, section 2b, storing values and memory)

Apply functions

- It's a “family” of functions (tapply, lapply, etc)
- R's own way of performing iterations
- The focus is not just on **speed**, but on **readability** and **clarity**.
- We are always trying to write code that is easy to read and maintain!
- First though, it has to work.

Apply functions

- They are a family of functions (tapply, lapply, etc)
- R's own way of performing iterations
- The focus is not just on **speed**, but on **readability** and **clarity**.
- We are always trying to write code that is easy to read and maintain!
- First though, it has to work.

This is seriously one of the pillars of writing code:

Premature optimization is dangerous!

Apply functions

- We will see a few examples using both apply() and sapply().
- Apply syntax: `apply(X, MARGIN, FUN,...)`
- Sapply syntax: `sapply(X, FUN,...)`
- The help is very useful for the syntax of these functions and to see what they do!
- Open R script, section 3) apply.

End of part 1 exercise

- In Exercise 1, you will try to “link together” all that we have seen till now by trying to do the same action in 3 different ways
  - Via R's vectorization
  - Via **one of the** apply() functions
  - Via a for loop
- It is a big task but don't be afraid to ask around!

Control flow : conditional executions

- Using ‘if’ alone, you need:
  - A condition, e.g “ if x > y “
  - Commands to apply if the condition is fulfilled, i.e. if x is indeed > than y.

Comparison Operators

equal: ==
not equal: !=
greater/less than: > <
greater/less than or equal: >= <=

Logical Operators

and: &
or: |
not: !

Control flow : conditional executions

- Using ‘if’ alone, you need:
  - A condition, e.g “ if x > y “
  - Commands to apply if the condition is fulfilled, i.e. if x is indeed > than y.
- Syntax:

```
if (condition)
{
   command(s)
}
```

When the condition is not met, R simply skips the commands between {}

➔ The cases where the condition is not fulfilled remain unchanged

Control flow : conditional executions

- Using ‘if else’
  - A condition + commands
  - Alternative commands to apply if the condition is not met
- Syntax:

```
if (condition)
{
   command(s)
}
```

Control flow : conditional executions

- Using ‘if else’
  - A condition + commands
  - Alternative commands to apply if the condition is not met
- Syntax:

```
if (condition)
{
   command(s)
} else {
      altenative commands
        }
```

Important to keep this on the same line!

Control flow: combine all!

- Possible and quite common to integrate conditions in ‘for’ loops:

```
for (i in 1:n)
{
  if (condition1)
  {
    do this
  } else
    {
    do that
    }
}
```

Handy, but can be very slow, and quite tricky.

Control flow: combine all!

- Possible and quite common to integrate conditions in ‘for’ loops:

```
for (i in 1:n)
{ # beginning of for loop
  if (condition1)
  { # beginning of if condition
    do this
  } else  # end of if
    { # beginning of else
    do that
    } # end of else
} # end of for loop
```

Brackets everywhere!!

Control flow: combine all!

- Also possible to make nested loops:

```
for (i in 1:n)
{
  for (j in site.list)
  {
    …
  }
}
```

- …. But much better to use vectoisation and the apply() family!

Example

- Created a classification of the sepal length column from the iris dataframe: if the length is <= mean, the class is “Small” and if the length is >mean, the class is “Large”

Exercise

- Create another classification using petal length : if the length is <=2, the class is “Small”, if the length is >2 and <=6, the class is “Large” and if the length >6 the class is “Extra Large”.

Debugging ‘for’ loops

- Check your brackets!! And comas, typos…
- Give values to ‘i’ and run step by step, each time checking the values in each variable
- If you get an error message that signals the loop did not go through completely, check value of ‘i’, it will tell you where things started not working.

Built-in functions

- R has a variety of “built-in” functions
  - read.table, mean, apply,… etc
- You can ‘unfold’ most R functions to see how they are coded: lots of ‘for’s and’if’s!
- Usually part of packages, which anyone can download, install, and use

  ➔R is open source. You can write your own package and share it too (!)

Built-in functions: help menu

- ?read.table
- {utils} : from package called ‘utils’
- ‘Usage’ section: tells you what all the possible **arguments** (=inputs) are, and what their **default value** is. Don’t need to precise them all, but good to know you have the option to.
- ‘Arguments’ section: tells you what the arguments actually mean, and what other non-default options are. Not always straight-forward.
- ‘Details’: More precisions about methods to use.

Built-in functions: help menu

- ‘Value’ = **output**, result. Lists all the objects you can access if you call them properly.
- Authors and references, very useful.
- See also
- **Examples:** copy-paste them in a new script and run them: best way to understand how functions work, and their potential.

Writing functions

- **Why??**
  - It's easy (you'll see)
  - Faster
  - Cleaner, both for you and the people you might share your script with
  - "every task you will be doing more than once… should be written as a function" (supervisor quote).

Writing functions

- **How??**
- You need: argument(s), commands, output(s)
- Syntax:

```
name.function <- function (arguments)
{
        commands
        return (result)
}
```

- Now.. Practice!