Programming Skills: more R fun(ctionality)

Modern Techniques in Modelling

Introduction

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

What we will introduce in this session

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

- ‘control statements' to automate and increase functionality
- 'functions' in R and how to write your own
- 'packages' in R to use other people’s code
- 'sourcing’ other code from different files

Control Statements

Types of Control Statements

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

- ‘Loops’ are what they sound like, they repeatedly loop through a bit of code, evaluating it each time. Most programming languages have loops, let’s look at the two popular ones:
  - ‘for’ loops
  - ‘while’ loops

- Other helpful control statements are:
  - ‘if-else’ statements
  - ‘break’ / ‘next’ statements

For-loops

Let's take a look at an example:

```
square.vector <- c()
for (counter in 1:10){
  counter.square <- counter^2
  square.vector  <- c(square.vector,
                      counter.square)
}
```

What is this doing?

While-loops

Let's take a look at an example:

```
x <- 5; # set value of x
while (x > 0){
  x <- x - 1 # on every loop, minus 1 from x
  print('x is positive')
}
```

Be careful with 'while' loops…

If-Else statements

Let's take a look at an example:

```
if (x < 0){
  print('warning: x is negative')
} else{
  print('x is positive, carry on')
}
```

```
if (x < 0){
  print('warning: x is negative')
}
```

'if' does not need to be followed by 'else'

Break statement

An example:

```
square.vector <- c()
for (counter in 1:10){
  counter.square <- counter^2
  if (counter.square > 80){
    break
  }
  square.vector <- c(square.vector,
                     counter.square)
}
```

What will `square.vector` equal when we run this?

What will `counter` equal when we run this?

Nested Statements

- You can add as many statements to your code as you like

Nested Statements

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

An example:

```
# initialise your variables
index.mort      <- 0
index.ps        <- 0
mortality.rate <- matrix(,nrow=3,ncol=3)

# Loop around the possible values for mort
for (mort in c(103, 401, 322)){
  index.mort <- index.mort + 1
  index.ps   <- 0

  # Loop around the possible values for pop size
  for (ps in c(1e4, 5e4, 7.5e4)){
    index.ps <- index.ps + 1

    mortality.rate[index.mort,index.ps] <-
      10000 * mort / ps
  }
}
```

Loop alternatives

Sometimes loops are difficult to read, difficult to write, and take up a lot of lines of code

Often there are multiple ways of achieving the same goal in R – without loops.

THINK: what is a) quicker to run, b) easier to read

Let's rewrite our previous example:

```
# Define the number of deaths
mort <- c(103, 401, 322)
# Define the population size
ps <- c(1e4, 5e4, 7.5e4)
# Enumerate all the combinations of mort and ps
pop.epi <- expand.grid(mort.val = mort, ps.val = ps)
# Calculate the mortality rate per 10,000
mortality.rate <- 10000 * pop.epi$mort.val /
                          pop.epi$ps.val
```

Functions

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

What is an R function?

Any set of operations that, when given a set of arguments (or `NULL`), returns an object

AND

where the set of operations are enclosed within the `function{}` keyword

What is an R function?

```
# VaccineThreshold function takes two arguments
VaccineThreshold <- function(trans.rate,
recovery.rate){

  # equation for R0 in an SIR model
  R0 <- trans.rate / recovery.rate

  # equation for the critical vaccination
threshold
  vaccine.threshold <- 1 - 1/R0

  # output of the function
  return(vaccine.threshold)
}

VaccineThreshold(1, 0.2)
```

What is an R function?

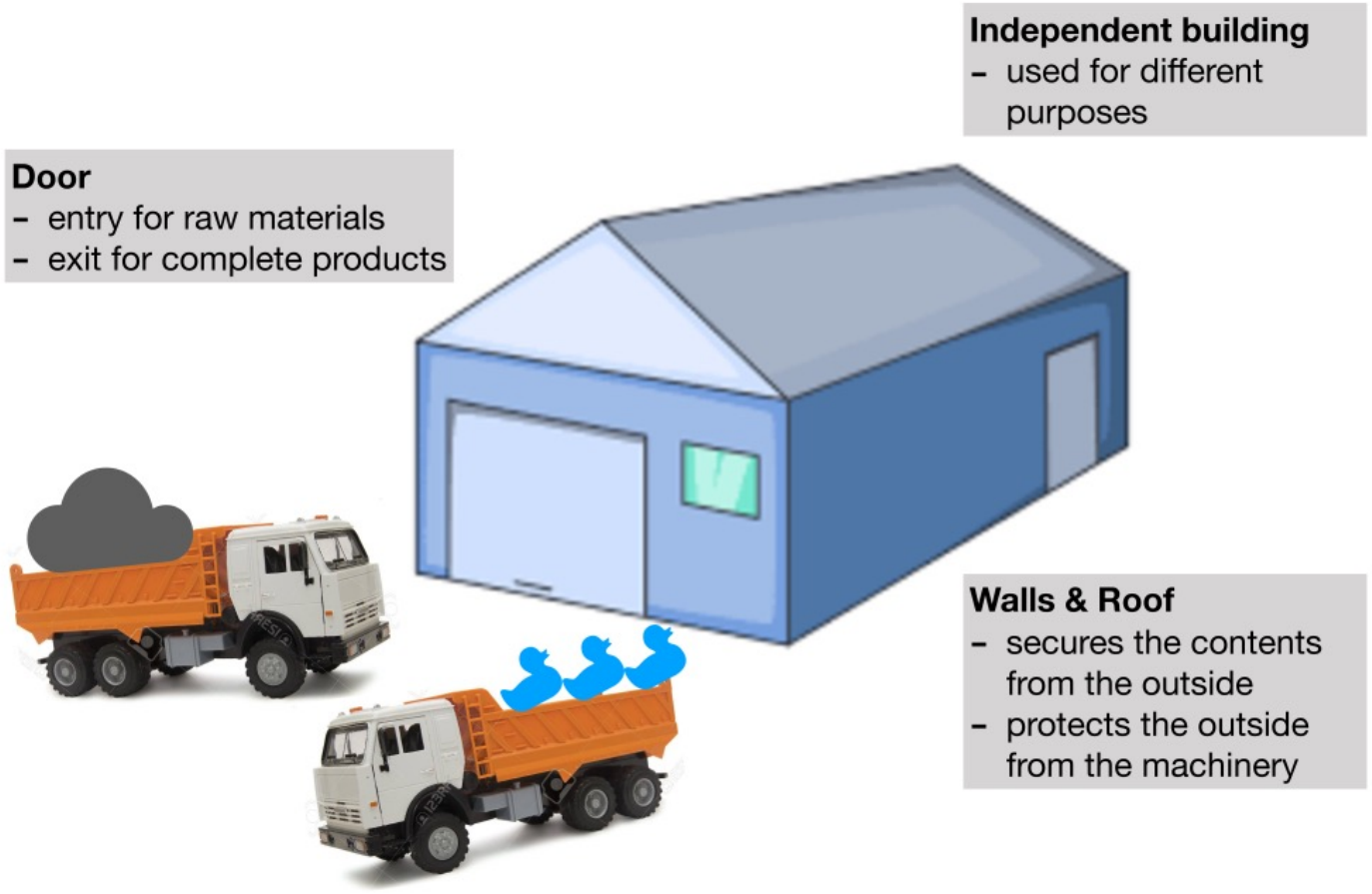
Compare this regular R `script':

```
#Example of an R script
a <- 1; b <- -4; c <- -2
sol <- c(0,0)
sol[1] <- (-b + sqrt(b^2 - 4*a*c))/(2*a)
sol[2] <- (-b - sqrt(b^2 - 4*a*c))/(2*a)
print(sol)
```

To this function:

```
#Example of an R function
quadratic.soln <- function(a,b,c){
  sol <- c(0,0)
  sol[1] <- (-b + sqrt(b^2 - 4*a*c))/(2*a)
  sol[2] <- (-b - sqrt(b^2 - 4*a*c))/(2*a)
  return(sol)
}
quadratic.soln(1,-4,-2)
quadratic.soln(b=-4,c=-2,a=1)
```

Why do we need to use functions?



An aside on Scoping

- Scoping is how R knows where to look for value assignments e.g. `print(a)` where does R look for `a`?

- R looks for `a` in its **current** working environment (e.g. function or top level workspace), if it can't find it, it looks in the **level above**, then the **next level above** etc.

- If it can't find it in any of these environments, R will throw an error

An aside on Scoping

– But why does it matter? Let's look at an example:

```
Reff.calc <- function(R0){
  reff <- R0 * imm.prop
}
```

– If we haven't defined `imm.prop`, there's an error (not bad), otherwise it might use a previously defined value that you might not expect, and you'll never know (very bad)

– You can find out what's in your working environment by typing `ls()` and remove a variable by `rm(imm.prop)`

Packages

- A package is a bundle of functions, already written and documented by another R user
- Often there is a package with functions already written to save you reinventing the wheel
- In Rstudio, either GUI: `Tools > Install Packages...` or in console type `install.packages("myPackage")`
- Now, when you want to use a package, simply type `library(myInstalledPackage)`

Sourcing code from different files

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

- You may want to split your code between multiple .R files
  - readibility: too much code for one document
  - organisation: group different functions into the same thematic files
  - error reduction: any replication of code writing WILL lead to errors (better to be lazy!)
- Simply write `source("myfilename.R")` into a script / function, and R will read in the contents of myfilename.R at the point where `source` is called
- You can use your knowledge of scoping to make sure you do this correctly!

Over to you

- Open up `Practical_P02_ProgrammingSkills.R`