# Week 1: Introduction to R

R content of ACTL1101

## Learning outcomes

By the end of this topic, you should be able to do the following in `R`:

- perform basic calculations
- perform relational and logical operations
- define variables
- recognise and create data of different types
- generate random observations

## Reference

These slides often make reference to specific pages from the following book:

- Lafaye de Micheaux P, Drouilhet R and Liquet B (2013), *The R Software: Fundamentals of Programming and Statistical Analysis*, Springer, New York

This book serves as the main (though not *only*) reference for the R content of ACTL1101. While you can buy a hard copy of this book at the UNSW bookshop, it is also downloadable for free at the following link (you may have to enter your UNSW credentials to access it).

## Why learn R?

- Made by statisticians for statisticians for **data analysis**
- Open source (**free!**)
- Relatively **simple**
- Great tool for **data visualisation**
- Can handle **large data sets**
- Pretty **fast** (and, if needed, efficiency can be increased with some 'tricks' like RCPP)
- Used in the **actuarial industry** (and others!)
- The majority of actuarial or statistical **textbooks** use `R`

- Used in **many other actuarial courses** at UNSW
- A wealth of **resources available**, e.g. Quick-R, R Seek, Stack Overflow, and many others

# R is a calculator

## Simple calculations - page 38

R can easily replace all the functionalities of a (sophisticated!) calculator.

```
sin(2*pi/3)      # <--- this symbol is for comments.
```

```
[1] 0.8660254
```

```
5^2              # Same as 5*5.
```

```
[1] 25
```

```
sqrt(4)          # Square root of 4.
```

```
[1] 2
```

```
log(1)           # Natural logarithm of 1.
```

```
[1] 0
```

```
c(1,2,3,4,5)     # Collection of the first 5 integers.
```

```
[1] 1 2 3 4 5
```

```
c(1,2,3,4,5)*2   # First five even numbers.
```

```
[1]  2  4  6  8 10
```

### R is a calculator - Exercise

Calculate the following

- $\sqrt[3]{8}$
- $e^2$

### R is a calculator - Solution

```
8^(1/3)
```

```
[1] 2
```

```
exp(2)
```

```
[1] 7.389056
```

## Storing values

### Page 39

R responds to your requests by displaying the result obtained after evaluation. However, this result is displayed, then lost.

To store values, one can use the assignment arrows: `<-` or `->`, or the more standard `=`.

```
x <- 1 # Assignment.
x # Display.
```

```
[1] 1
```

```
2 -> x # Assignment (in the other direction).
x # Display.
```

```
[1] 2
```

```r
x = 3 # Assignment.
x # Display.
```

```
[1] 3
```

```r
(x <- 1) # Assignment AND display.
```

```
[1] 1
```

These stored values are known as variables (more on these later).

### Assignment operators

While there are many ways to assign variables in `R`, it is recommended that you either use `=` or `<-`.

Some of you may also be asking whether there are any differences between the different assignment operators, and there are, but they are unlikely to have any effect on the kind of things we will do in this course.

If you are interested in the differences anyway, see this link for more information.

Once we move to `RStudio`, there is an inbuilt shortcut Alt + - for typing `<-`.

### Variables - Exercise

Perform the following tasks

- Create a variable `var1` whose value is the sum of $\{1, 2, 3, 4\}$
- Create a variable `var2` whose value is the multiplication of $e$ and $\pi$
- Create a variable `var3` whose value is the sum of `var1` and `var2`
- Display the value of `var3`

### Variables - Solution

```r
var1 <- sum(c(1,2,3,4))
var2 <- exp(1)*pi
var3 <- var1+var2
var3
```

```
[1] 18.53973
```

### Assignment operations - Exercise

Which of the following are valid methods of assigning `x` with the value 2?

- `x = 2`? Yes
- `2 -> x`? Yes
- `x -> 2`? No

- `x <- 2`? Yes
- `2 = x`? No
- `2 <- x`? No

## Vectors

### Page 51

- Vectors are a very important type of **data structure** in `R`. We will see other types of data structures in Week 2, but for now we concentrate on vectors.
- A vector is a **sequence of data points** of the same type.
- You can create a vector in different ways. For instance, the function `c()` produces a vector.
- Operations performed on vectors are done **element by element**.

```
c(1,2,3)
```

```
[1] 1 2 3
```

```
c(1,2,3) + c(4,5,6)
```

```
[1] 5 7 9
```

```
c(1,2,3) * c(4,5,6)
```

```
[1]  4 10 18
```

### Vectors (continued)

- To produce a vector, you can also use function `seq()`

```
seq(from=0, to=1, by=0.1)
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
seq(from=0, to=20, length=5)
```

```
[1]  0  5 10 15 20
```

- Or simply use the colon :

```
vec <- 2:10
vec
```

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

### Vectors - Exercise - Page 73

- Create a vector of numbers from 4 up to 5, where the increment is by 0.3

```
seq(from=4, to=5, by=0.3)
```

```
[1] 4.0 4.3 4.6 4.9
```

- Create a vector of numbers equally spaced from 4 to 5, where the total length is 5

```
seq(from=4, to=5, length=5)
```

```
[1] 4.00 4.25 4.50 4.75 5.00
```

## Relational operations

### Pages 97-98

You can perform relational operations in `R`, which will output logical values (`TRUE`/`FALSE`) (note: you can scroll down)

```r
# Strictly greater
2 < 3
```

```
[1] TRUE
```

```r
2 < 2
```

```
[1] FALSE
```

```r
# Greater or equal
2 <= 2
```

```
[1] TRUE
```

```r
# Equal
1 == 1
```

```
[1] TRUE
```

```r
# Not equal
1 != 1
```

```
[1] FALSE
```

```r
# Vector of relational operations
c(2>1,4>2,pi==3)
```

```
[1]  TRUE  TRUE FALSE
```

```r
# Equality between two vectors
c(1,2,3)==c(1,1,3)
```

```
[1]  TRUE FALSE  TRUE
```

Note that in the last case, since two vectors are compared, three results are given, one for each individual relation.

# Logical operations

## The AND operator - Pages 97-98

You can perform logical operations in `R`, which will output logical values. Any logical operation takes the form

`statement1 OPERATOR statement2`

We start with the operator `AND`. If `statement1` is `T` **and** `statement2` is `T`, then `AND` outputs `T`, otherwise, it outputs `F`.

In `R` there are two `AND` operators, `&` and `&&`. We start with `&` which is an element-wise comparison.

It compares the first element of the first vector to the first element of the second vector, then the second of the first vector to the second of the second vector, etc.

It therefore returns a **vector of logical values**. See if you can understand how the output below is produced.

```
c(T,T,F,F) & c(T,F,T,F)
```

```
[1]  TRUE FALSE FALSE FALSE
```

Obviously, we can combine this with the previous section to construct statements like

```
c(2>1,4>2) & c(1>2,pi>=3)
```

```
[1] FALSE  TRUE
```

## The AND operator continued

The `&& AND` operator is known as the short-circuit evaluation

- It can only be used on **scalars**, not vectors
- It hence returns a **single** logical value
- Normally used in programming control flows (Week 3)

```
T && F && T
```

```
[1] FALSE
```

```
2 > 1 && 3 > 0 && 9 > 5
```

```
[1] TRUE
```

```
# if you try for example c(2>1,4>2)&&c(1>2,3>0), you get an error
```

So what is the point of this? Well 'short-circuit' means that if one condition fails, the entire condition fails and it does not check the other conditions. This means you can do things like

```
x <- -2
x > 0 && sqrt(x) < 20
```

```
[1] FALSE
```

because as soon as it finds that the first statement was false, it exits the operation rather than trying to take the square root of a negative number.

### The OR and NOT operators

The `OR` operator does exactly what it sounds like. If `statement1` is `T` **or** `statement2` is `T`, then it returns `T`, otherwise it returns `F`.

Again we have both `|` and `||`.

```
c(T, T, F, F) | c(T, F, T, F)
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

```
c(2 > 1, 3 < 1) | c(0 > 1, 4 < 1)
```

```
[1]  TRUE FALSE
```

```
0 > 2 || 3 < 4 || 1 < 0
```

```
[1] TRUE
```

The `NOT` operator is much simpler and represented by `!`; it just inverts the result, i.e. `T` becomes `F`, and `F` becomes `T`:

```
!c(T, T, F)
```

```
[1] FALSE FALSE  TRUE
```

```
!(2 > 3)
```

```
[1] TRUE
```

### Logical operations - Pages 97-98

There are many other functions in `R` that operate on logical values/vectors, such as:

```
any(c(F,T,F,F))
```

```
[1] TRUE
```

```
all(c(F,T,F,F))
```

```
[1] FALSE
```

Another useful trick is that internally `R` (and most programming languages) treat `T` as `1` and `F` as `0`, e.g.,

```
T+T+T+F+F
```

```
[1] 3
```

```
sum(c(T,F,T,F,F,F))
```

```
[1] 2
```

You can also do things like:

```
sum(c(1, 2, 3, 4) > 2.5) # Count number of "TRUE" in the vector
```

```
[1] 2
```

```r
mean(c(1, 2, 3, 4) > 2.5) # Calculates the proportion of TRUEs
```

```
[1] 0.5
```

```r
sum(c(F, T, F, F)) > 0 # This is equivalent to any()
```

```
[1] TRUE
```

```r
sum(c(T, T, F)) == 3 # This is equivalent to all()
```

```
[1] FALSE
```

# Variables in R

## Page 40

A variable is an object in `R`. There are rules for choosing a variable name:

- a variable name can only include **alphanumerical characters**, **underscore** (_) and the **dot** (.);
- variable names are **case sensitive**, which means that R distinguishes upper and lower case;
- a variable name **may not** include white space or start with a digit.

Note: use **meaningful** names for your variables to improve the readability of your code.

## Data types - page 50

One of the main strengths of `R` is its ability to organise data in a structured way. This will turn out to be very useful for many statistical procedures and data analysis.

| Data type | Type in R | Display |
|---|---|---|
| real number (integer or not) | numeric (double) | 3.27 |
| integer | numeric (integer) | 3 |
| complex number | complex | 3+2i |
| logical (true/false) | logical | TRUE or FALSE |
| missing | logical | NA |

| Data type | Type in R | Display |
|---|---|---|
| text (string) | character | “text” |

### Data types - Examples - pages 46-48

```
a <- 1
typeof(a)
```

```
[1] "double"
```

```
c <- as.integer(a)
typeof(c)
```

```
[1] "integer"
```

```
is.numeric(a)
```

```
[1] TRUE
```

```
is.integer(a)
```

```
[1] FALSE
```

```
x <- TRUE # same as: x <- T
typeof(x)
```

```
[1] "logical"
```

```
is.logical(x)
```

```
[1] TRUE
```

```
b <- 3.4
my.vec <- c(b>a,a==b)
typeof(my.vec)
```

```
[1] "logical"
```

## Data types - Missing data - pages 48-49

A missing or undefined value is indicated by the instruction NA (for Non Available).

```
x <- c(3,NA,6)
is.na(x)
```

```
[1] FALSE  TRUE FALSE
```

Normally if you try and do numerical operations with `NA`, the whole operation becomes `NA` as you can see below. This is done to alert you that their are `NA`s in your data, but if you want to ignore them, some functions (e.g. `mean` and `sum`) come with an `na.rm` argument

```
sum(x)
```

```
[1] NA
```

```
sum(x,na.rm=T)
```

```
[1] 9
```

```
mean(x,na.rm=T)
```

```
[1] 4.5
```

This argument removes all the `NA`s before applying the operation.

Dealing with `NA`s is **very** important once we get to importing and using external data.

## Data types - Character strings - page 49

Any information between quotation marks (single ' ' or double " ") corresponds to a character string. Try the following commands:

```
a <- "R is my friend"
typeof(a)
```

```
[1] "character"
```

```
is.character(a)
```

```
[1] TRUE
```

### Data types - Exercise

Given that

```
var1 <- as.integer(3)
var2 <- "1"
```

- Determine the types of `var1` and `var2`

```
typeof(var1)
```

```
[1] "integer"
```

```
typeof(var2)
```

```
[1] "character"
```

- Convert `var1` to a double precision number using `as.double`

Notice that `as.double` still needs to be re-assigned to the variable, otherwise it does nothing. Most functions in `R` behave this way and do not perform "in-place" changes

```
as.double(var1)
```

```
[1] 3
```

```
typeof(var1) # The above as.double did not change var1
```

```
[1] "integer"
```

```
var1 <- as.double(var1)
typeof(var1)
```

```
[1] "double"
```

## Probability distributions and random variables in R

### Distribution-related functions

- We have seen in the 'Probability' theory part of this coure, that **random variables** are mathematical descriptions of the random phenomena encountered in everyday life.
- Many probability distributions are implemented in base `R`. There are typically four functions you can use for each distribution. For the **Normal distribution** they are `dnorm` (density function), `pnorm` (CDF), `qnorm` (quantile function) and `rnorm` (random generator).

```
dnorm(1.96,  mean = 0 , sd = 1)
```

```
[1] 0.05844094
```

```
pnorm(1.96,  mean = 0 , sd = 1)
```

```
[1] 0.9750021
```

```
qnorm(0.5,   mean = 0 , sd = 1)
```

```
[1] 0
```

```
qnorm(0.975, mean = 0 , sd = 1)
```

```
[1] 1.959964
```

```
rnorm(1, mean = 0, sd = 1)
```

```
[1] 0.07982161
```

- Many more probability distributions are implemented in `R`. For instance, can you guess what `dexp()` does? Or `pbinom()`?

### Generating random variables in R - Page 74

- To conduct statistical analysis, it is often useful to be able to generate realisations from random variables, and `R` is an apt tool to do so throught the use of functions `runif()`, `rnorm()`, `rgamma()`, etc…

```
# Generate two random obervations from a Uniform[0,1]
runif(n = 2)
```

```
[1] 0.6120920 0.6383312
```

```
# Generate four random observations from a Uniform[2,7]
runif(n = 4, min=2, max=7)
```

```
[1] 2.755362 2.610214 5.344028 6.199502
```

### Generate random variables in R - Exercise - Page 74

- Generate two realisations of a standard normal random variable

```
rnorm(2)
```

```
[1] 0.03739676 0.19007710
```

- Generate one realisation of a normal random variable with mean 10 and standard deviation 0.1

```
rnorm(1, mean=10, sd=0.1)
```

```
[1] 10.0611
```

## Homework

We place below some problems for you to solve as extra practice… try them out!

### Homework Exercises 1

What is the output produced by the following `R` codes? Try to predict it before typing the commands in `R`!

- `1:3^2`
- `(1:5)*2`
- `root.of.four <- sqrt(4)`
- `TRUE + T +FALSE*F + T*FALSE + F`

(Inspired from Exercises 3.1-3.13 in the `R` Textbook.)

### Homework Exercise 2

What can be improved about the following variable names? Suggest a better alternative.

- `delaytime`
- `the_number_of_marks_higher_than_50`
- `20_day_limit`
- `child/adult`
- `pi`
- `variable1`

### Homework exercise 3

Create a vector `x` consisting of 10 randomly generated Binomial($n = 100, p = 0.1$) variables, and display the result.

### Homework exercise 4

Create 3 randomly generated standard Normal random variables.

### Homework exercise 5

By generating an independent sample of size 10000, approximate the probability that a standard normal random variable is greater than 1.96

Hint: check slide 24 if you are having trouble with calculating the probability after simulating.