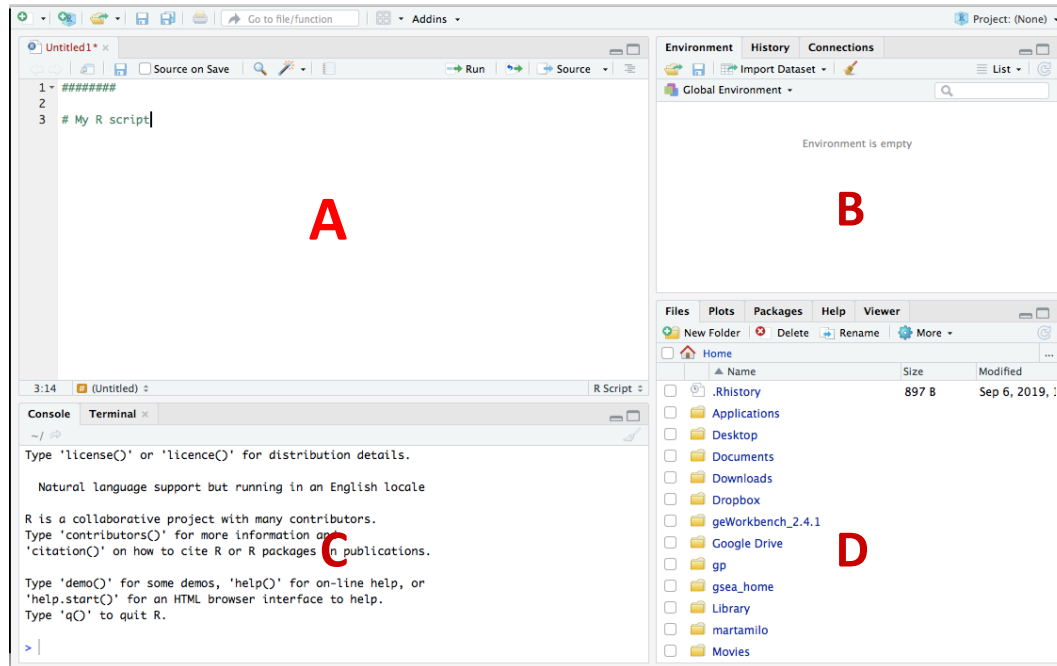


# Introduction to Data Handling in R (BMS 109) Week3

# Today you will learn

- Functions in R
- Handling R objects ( variables, arrays etc..)
- Set arrays of values and access them
- Set matrix and tables
- Manipulating tables and matrix

# Quick recap of R Studio



A :Script Editor

B :Global Environment and History

C: Console

D :Everything else (e.g. file browser)

1. Open up RStudio All Programs > Rstudio
2. Maximise the RStudio window (always do this!)
3. Open last week script

## Print the values of objects

```
x <-3
y <-10
z <-15

s <-x+y+z

d <-(x-y)/z

m <-x*y*z
v <-s^4
```

s  
cat("Ratio: ", d, "\n")  
print(paste("Multiplication: ", m))  
print(paste("Power of 4: \n", v))  
print(sq)

**Write these commands in the newly opened script and use comments to describe what you are doing.**

```
> s
[1] 28
> cat("Ratio: ", d, "\n")
Ratio:  -0.4666667
> print(paste("Multiplication: ", m))
[1] "Multiplication:  450"
> print(paste("Power of 4: \n", v))
[1] "Power of 4: \n 614656"
> print(sq)
[1] 3.464102
```

# Functions

A function in R is very much like a mathematical function. They (usually) takes one or more inputs, does something to them and then returns the result to us.

```
my_func_name(arg1 = val1, arg2 = val2, ...)
```

my\_func\_name: the name of the function (e.g. print)

arg1: name of the first argument

val1: value of the first argument

A function may have 0, 1, or more "arguments". The arguments are the inputs (name value pairs) to the function. Remember: name on the left, value of the right.

# Examples of using functions

*Variables* can contain collections of letters called **strings**. We manipulate strings differently from how we manipulate numbers. We use commands such as `paste`. For example:

```
myname <- "Marta"
Greeting <- "Ciao"

#Let's join these together using R's paste function.
#Sep determines the type of separation you want.

message <- paste(Greeting,myname,sep=" ")
print(message) # Print out the message

[1] "Ciao Marta"
```

## Exercise

Write code to print your name, your email and the module code separated by a comma.

# Examples of using functions

If we want to round a number we can ask R to do this. The function we would use is `round( )`

Round needs arguments, we start with one. We give it the value 3.141593.

```
> round(x = 3.141593)
[1] 3
>
> x
Error: object 'x' not found
```

We can also ask R for help and describe here all information about the function.

```
> ?round
round(x, digits = 0)
```

The description appears on the bottom right panel

```
round(3.141593,2)
[1] 3.14
```

You do not have to name arguments in R – it will match them up by position.

# Exercise

Using functions experiment with the following functions, use the help “?” to convince yourself you know what they do. Add comments to explain your findings:

`sqrt`  
`log`  
`sum`  
`seq`  
`c`  
`round`

Hint: Don't just use whole numbers.



# Vectors

Vectors are the simplest type of "object" in R— It can be made of one or more elements ( values). Can be numerals or characters, or strings.

We use them to associate a row of values to the same name.

When we want to extract single element of the vectors we use [ ] and their position.

```
> x<-2  
> x<-c(2,3)  
> x  
[1] 2 3
```

Now let's take a look at what the symbol :

**It is used in to build a sequence of number and it is frequently used in vectors.**

```
> 1:15  
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

We have build a sequence of 15 numbers

## Vectors (cont ...)

You can assign a set of values to an object and this can be in the form of a row of values (vector) or a table (matrix). For example to build a vector with numbers from 1 to 10 we can use any of the following methods:

```
x <- 1:10  
x <- seq(1,10,by=1)  
x <- seq(length=10,from=1,by=1)  
x <- c(1,2,3,4,5,6,7,8,9,10)    # c = concatenate
```

You can generate random sequences of number using commands like `sample()`, `runif()` and `rnorm()`.

The first one just creates a vector of random numbers

The second function creates a vector of uniformly distributed random numbers

The third function creates a vector of normally distributed random numbers

The function `length(x)` gives you the length of the vector It is very useful,

## Vectors (cont...)

### Exercise:

Explore the sampling functions `sample()`, `runif()` and `rnorm()` to create a sequence of 10 random integers from 1:100. Use the comments to describe briefly what you are doing and assign the outputs of the functions to variables `x`, `y`, `z` respectively. For example you can use

```
x<-sample(1:100, 10, replace=TRUE)
y<-runif(10,min=1,max=10)
z<- rnorm(10,mean=0,sd=1)
x
y
z

> x
[1] 64 97 36 35 82 25 45 92 6 8
> y
[1] 2.817598 9.572927 5.784630 4.687285 7.461036 1.305383 4.009328
8.185989 2.789331 2.984100
> z
[1] -0.2387676 0.6077371 -0.2714998 -1.1619429 -0.5161393
-0.2406907 -1.5376712 -0.8294226
[9] -1.4247751 0.8918644
```

## Vectors (cont...)

We access elements of a vector using the [ ] brackets and the position of the elements we want.

For example if we want the fourth element of x we write:

```
> x[4]  
[1] 52
```

### Exercise

Get the first, the second and the forth element of y

```
> x[c(1,2,4)]  
[1] 9 27 52
```

Get the even elements of z ( hints: you can write a vector of even numbers in 1:10 or use the seq(2,10) specify by=2

```
> index<- seq(2,10,by=2)  
> z[index]  
[1] -0.2387676 -0.2714998 -0.5161393 -1.5376712  
-1.4247751
```

# Matrix and tables

Often we need to organise our data in a table with rows and columns. Vectors of rows and columns are called matrix.

We more often use the command `matrix()`, to create a matrix or rearrange a set of data. `matrix()` needs the data, `nrow`, `ncol`. For example:

```
M<- matrix( x, nrow = 2, ncol = 5)
```

```
M
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	64	36	82	45	6
[2,]	97	35	25	92	8

We can index the elements of `M` in the same way we used for vectors. The only difference: now we need two indices in the square brackets `[ , ]`, because `M` is two-dimensional. The first index corresponds to the rows, the second to the columns.

## Matrix and tables(cont...)

Create new matrix called M1 of 20 elements with nrow=4, ncol=5

```
M1<- matrix(1:20,nrow=4,ncol=5)
```

M1

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	5	9	13	17
[2,]	2	6	10	14	18
[3,]	3	7	11	15	19
[4,]	4	8	12	16	20

We access elements of the matrix specifying the two indices. For example to the element on second row and third column

M1[2,3]

We can also access more than one element with the function c()

```
> M1[c(1,3,4),4]  
[1] 13 15 16
```

## Matrix and tables(cont...)

We also can skip a whole column and/or row by using the index with the minus sign. For example if we want to skip the third column we can write

```
> M2<- M1[, -3]
M2
```

	[ ,1]	[ ,2]	[ ,3]	[ ,4]
[ 1, ]	1	5	13	17
[ 2, ]	2	6	14	18
[ 3, ]	3	7	15	19
[ 4, ]	4	8	16	20

What are the dimensions of M2?

Use `dim(M2)` to discover them. Very useful function.

```
> dim(M2)
[1] 4 4
```

## Exercise

Create from the matrix M1 new sub-matrices using the instruction below. Assign new names to each sub-matrix and find the their dimensions:

- The 2x2 matrix forming the first two row and the first two column
- The first two rows with all columns
- Second column missing
- The first two row and the 4 with third column missing



# Solutions

- The 2x2 matrix forming the first two row and the first two column

```
> M3<-M1[1:2,1:2]  
> dim(M3)  
[1] 2 2
```

- The first two rows with all columns

```
> M4<- M1[1:2,]  
> dim(M4)  
[1] 2 5
```

- Second column missing

```
> M5<- M1[-2,]  
> dim(M5)  
[1] 3 5
```

- The first two row and the 4 with third column missing

```
> M6<-M1[c(1,2,4),-3]  
> dim(M6)  
[1] 3 4
```

# Before you go

Comment well your script.

Save your .R file

Close Rstudio

Log off