Module 1 - The R Programming Environment

1 - Setting up Swirl

This is just a quick guide of how one can submit ones code to coursera.

2 - Basic Building Blocks

R can be used as a simple calculator using +, -, * and /

```
5 + 7
```

```
## [1] 12
```

A simpler method is to define elements as a variable.

```
x <- 5+7
```

You can view a variable by typing its name

```
x
```

```
## [1] 12
```

Now im storing the variable y as x-3

```
y <- x-3
```

We will now create a vector with the c() function.

```
z <- c(1.1, 9, 3.14)
```

We can view the results of different variables inside other functions

```
c(z, 555, z)
```

```
## [1]   1.10   9.00   3.14 555.00   1.10   9.00   3.14
```

We can take the square root of something like this

```
my_sqrt <- sqrt(z - 1)
```

New element called my_div is created, which is the result of z/my_sqrt. In this command R takes the three numbers present in z and divides each of them with the content of my_sqrt. The first number would then be 1.1/3.48, which equals 0.316.

```
my_div <- z/my_sqrt
```

If the number of elements dont match in each group R recycles the shorter vector. We can see an example of this here:

```
c(1, 2, 3, 4) + c(0, 10)
```

```
## [1]  1 12  3 14
```

If the shorter vector does not divide evenly into the recyceling function the following occurs.

```
c(1, 2, 3, 4) + c(0, 10, 100)
```

```
## Warning in c(1, 2, 3, 4) + c(0, 10, 100): longer object length is not a multiple
## of shorter object length
```

```
## [1]   1  12 103   4
```

3 - Sequences of Numbers

In this section we will create sequences of numbers in R.

In its simplest term, the easiest way to create a sequence of numbers in R is to use the : operator likes this:

```
1:20
```

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

Alternativly you can create "real" numbers using this:

```
pi:10
```

```
## [1] 3.141593 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593
```

You can also move the other way like this:

```
15:1
```

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

Tip: If you want documentation for a R operator you can do so like this:

```
?":"
```

```
## starting httpd help server ... done
```

An alternative to : is the seq() function. This function gives more control. Can be used like this:

```
seq(0, 10, by=0.5)
```

```
##  [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
## [16]  7.5  8.0  8.5  9.0  9.5 10.0
```

We make a sequence and store it as my_seq

```
my_seq <- seq(5, 10, length=30)
```

We can confirm the length of my_seq by:

```
length(my_seq)
```

```
## [1] 30
```

We can also use our variables to define other things, as for example we want a sequence of 1, 2, 3 with the same length of my_seq.

```
## This can be done using ##
1:length(my_seq)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```
## or ##
seq(along.with = my_seq)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```
## or (this is the best) ##
seq_along(my_seq)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

These built in functions are often more optimal to use when possible as they are optimized for a certain task.

Another function we can use is rep(), which repeates stuff.

```
rep(0, times = 40)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0
```

Or like this:

```
rep(c(0, 1, 2), times=10)
```

```
##  [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

or like this

```
rep(c(0, 1, 2), each = 10)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

4 - Vectors

There are two different types of vectors, either atomic or lists. Atomic vectors contains only one type of data while lists contain several types of data.This first part is based on atomic vectors.

Example of atomic vector

Numeric vector: 1, 2, 3, 4

Logical vector: TRUE, FALSE, NA (not avaliabe)

Character vector: “a”, “b”

We can create logical by using logical “conditions”.

```
num_vect <- c(0.5, 55, -10, 6)
tf <- num_vect < 1
```

We can altenativly do:

```
num_vect >= 6
```

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

We can use the following logical operators to get these expressions

< = Less than

= More than

== = identical

!= = inequality

<= Less or equal

=> More or equal

We can also add other stuff to this.

= at least one is t

& = Both is true

! = oposite

We will now work on character vectors, which are defined by "".

```
my_char <- c("My", "name", "is")
```

We can also combine different variables using the paste function to combine stuff.

```
paste(my_char, collapse = " ")
```

```
## [1] "My name is"
```

We can also add stuff to a vector

```
my_name <- c(my_char, "Anders")
```

Here we combine the paste function with my name.

```
paste(my_name, collapse = " ")
```

```
## [1] "My name is Anders"
```

We can also combine stuff like this

```
paste("Hello", "world!", sep = " ")
```

```
## [1] "Hello world!"
```

```
paste(1:3,c("X", "Y", "Z"), sep = "")
```

```
## [1] "1X" "2Y" "3Z"
```

Here is a example where there is recyceling.

```
paste(LETTERS, 1:4, sep = "-")
```

```
##  [1] "A-1" "B-2" "C-3" "D-4" "E-1" "F-2" "G-3" "H-4" "I-1" "J-2" "K-3" "L-4"
## [13] "M-1" "N-2" "O-3" "P-4" "Q-1" "R-2" "S-3" "T-4" "U-1" "V-2" "W-3" "X-4"
## [25] "Y-1" "Z-2"
```

4 - Missing values

dadasdas