Lesson 04 - Getting Started with R (notes)

PUT YOUR NAME HERE

Creating objects in R

```
weight_kg <- 55
```

When assigning a value to an object, R does not print anything. You can force R to print the value by using parentheses or by typing the object name:

```
weight_kg <- 55
(weight_kg <- 55)
```

```
## [1] 55
```

```
weight_kg
```

```
## [1] 55
```

For instance, we may want to convert this weight into pounds (weight in pounds is 2.2 times the weight in kg):

```
2.2 * weight_kg
```

```
## [1] 121
```

We can also change an object's value by assigning it a new one:

```
weight_kg <- 57.5
2.2 * weight_kg
```

```
## [1] 126.5
```

For example, let's store the animal's weight in pounds in a new object, `weight_lb`:

```
weight_lb <- 2.2 * weight_kg
```

and then change `weight_kg` to 100.

```
weight_kg <- 100
```

R executes code in top-down order. So what happens on line 10 occurs before line 11. What do you think is the current content of the object `weight_lb`? 126.5 or 220?

> write your answer here

Functions and their arguments

```
sqrt(4)
```

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

Let's look into the `round` function.

```
round(3.14159)
```

```
## [1] 3
```

We see that if we want a different number of digits, we cantype `digits = 2` or however many we want.

```
round(3.14159, digits = 2)
```

```
## [1] 3.14
```

If you provide the arguments in the exact same order as they are defined you don't have to name them:

```
round(3.14159, 2)
```

```
## [1] 3.14
```

And if you do name the arguments, you can switch their order:

```
round(digits = 2, x = 3.14159)
```

```
## [1] 3.14
```

Data Types

R objects come in different data types.

Numbers

```
im_a_number <- 50
im_a_number*2
```

```
## [1] 100
```

Letters

```
(im_a_character <- "dog")
```

```
## [1] "dog"
```

```
im_a_character*2
```

```
## Error in im_a_character * 2: non-numeric argument to binary operator
```

Boolean

```
3>4
```

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

```
sqrt(4)==2
```

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

Data Structures

Vectors

For example we can create a vector of animal weights and assign it to a new object `weight_g`:

```
(weight_g <- c(50, 60, 65, 82))
```

```
## [1] 50 60 65 82
```

A vector can also contain characters:

```
(animals <- c("mouse", "rat", "dog"))
```

```
## [1] "mouse" "rat"   "dog"
```

You can use the function `class()` to see what data type a vector is.

```r
class(weight_g)
```

```
## [1] "numeric"
```

```r
class(animals)
```

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

If you combine letters and numbers, everything will be treated as a character.

```r
(mix_match <- c(weight_g, animals))
```

```
## [1] "50"    "60"    "65"    "82"    "mouse" "rat"   "dog"
```

```r
class(mix_match)
```

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

Doing math on vectors

You can perform math operations on the elements of a vector such as

```r
weight_KG <- weight_g/1000
weight_KG
```

```
## [1] 0.050 0.060 0.065 0.082
```

When adding two vectors together, the elements in the same position are added to each other. So element 1 in the vector `a` is added to element 1 in vector `b`.

```r
a <- c(1,2,3)
b <- c(6,7,8)
a+b
```

```
## [1]  7  9 11
```

More complex calculations can be performed on multiple vectors.

```r
wt_lb <- c(155, 135, 90)
ht_in <- c(72, 64, 50)
bmi <- 703*wt_lb / ht_in^2
bmi
```

```
## [1] 21.01948 23.17017 25.30800
```

All these operations on vectors behave the same way when dealing with variables in a data set (data.frame).

If you want to add the values *within* a vector, you use functions such as `sum()`, `max()` and `mean()`

```
sum(a)
```

```
## [1] 6
```

```
max(b)
```

```
## [1] 8
```

```
mean(a+b)
```

```
## [1] 9
```

Subsetting vectors

If we want to extract one or several values from a vector, we must provide one or several indices in square brackets. For instance:

```
animals <- c("mouse", "rat", "dog", "cat")
animals[2]
```

```
## [1] "rat"
```

```
animals[c(2, 3)]
```

```
## [1] "rat" "dog"
```

The number in the indices indicates which element to extract. For example we can extract the 3rd element in `weight_g` by typing

```
weight_g[3]
```

```
## [1] 65
```

Conditional subsetting

Typically, these logical vectors are not typed by hand, but are the output of other functions or logical tests such as:

```
weight_g > 50
```

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

For instance, if you wanted to select only the values where weight in grams is above 50 we would type:

```
weight_g[weight_g > 50]
```

```
## [1] 60 65 82
```

You can combine multiple tests using `&` (both conditions are true, AND) or `|` (at least one of the conditions is true, OR):

Weight is less than 30g or greater than 60g

```
weight_g[weight_g < 30 | weight_g > 60]
```

```
## [1] 65 82
```

Weight is between 60 and 80lbs

```
weight_g[weight_g >= 60 & weight_g <= 80]
```

```
## [1] 60 65
```

The function `%in%` allows you to test if any of the elements of a search vector are found:

```
animals <- c("mouse", "rat", "dog", "cat")
animals[animals == "cat" | animals == "rat"] # returns both rat and cat
```

```
## [1] "rat" "cat"
```

```
animals %in% c("rat", "cat", "dog", "duck", "goat")
```

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

```
animals[animals %in% c("rat", "cat", "dog", "duck", "goat")]
```

```
## [1] "rat" "dog" "cat"
```

Order matters.

When considering string or character vectors or data elements, R treats everything in alphabetical order.

```
"four" > "five"
```

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

Data Frames

We can load the `diamonds` data set into our global environment by typing

```
diamonds <- ggplot2::diamonds
```

We can get an idea of the structure of the data frame including variable names and types by using the `str` function,

```r
str(diamonds)
```

```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Inspecting data.frame objects

Identifying variables

Below is an example of finding the average price for all diamonds in the data set.

```r
mean(diamonds$price)
```

```
## [1] 3932.8
```

Here is an example of finding the average price for `Good` quality diamonds.

```r
mean(diamonds$price[diamonds$cut=="Good"])
```

```
## [1] 3928.864
```