R Workshop

Lecture 1: Basics of Programming in R

1 How to print stuff

There are actually a few ways in which you can do this in **R**, but I will discuss only one way that is probably easy to remember. As you become more proficient in R, you can figure out the other ways :).
Note: Anything after "#" on a line is ignored. So, if it very useful for commenting.

```
#Printing something
#Don't forget the quotes, if it is a string.
print("Hello World!")

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

2 Assignment in R

There are at least two ways in which assignment is done in R.

1. The traditional way (the prescriptivists love it)
   x <- 1
2. The other way (the prescriptivists hate it; but I love it)
   x = 1

I will use the second way throughout. But, this comes down to individual preference. So, feel free to use the other one if you are more used to it. One important thing: be consistent about your choice so that you don't confuse yourself.

Note: For most purposes, both work the same way. You can read about some subtle difference about the scope of operation amongst other things on Stack Overflow.

3 Some useful datatypes in R

3.1 `vectors` (or Arrays)

A `vector` is a collection of similar elements (integers, decimal point numbers, characters, factors, ...). Below I have created different `vectors` with just a single value each.

```
#Integer vector
x = c(4)
print(x)    #Note: No need for quotes if it is a variable or if it is not a string.

## [1] 4

#Character vector
y = c("a")
print(y)

## [1] "a"
```

Now, we can see more complex `vectors` with multiple elements:

```
#Integer vector
x = c(5,6,7,8)
print(x)

## [1] 5 6 7 8

#Character vector
y = c("a","b","c","d")
print(y)

## [1] "a" "b" "c" "d"
```

3.1.1 Understanding `vector` math (and `vector` operations more generally)

R works off vectors, such that almost all of the functions naturally work with vectors. Let's work with addition and subtraction.

Create two new integer `vectors`:

```
#Integer vectors
x = c(5,6,7,8)
y = c(1,2,3,4)  #Also try, y = c(1:4)
```

Now, try to add the `vectors`:

```
x + y

## [1]  6  8 10 12
```

If the `vectors` are of different lengths, then R throws a "warning". But, it automatically cycles through the shorter `vector` so that they are the same length.

```
#Integer vectors
x = c(5,6,7)
y = c(1,2,3,4)
x + y

## Warning in x + y: longer object length is not a multiple of shorter object length

## [1]  6  8 10  9
```

3.2 data.frame

A `data.frame` is a collection of `vectors`, where each column is a different `vector`. For example, we can take the above `vectors` and make them a `data.frame`.

```
#Integer vectors
x = c(5,6,7,8)
y = c(1,2,3,4)

#Making a data.frame
data = data.frame(x, y)
print(data)
```

```
##   x y
## 1 5 1
## 2 6 2
## 3 7 3
## 4 8 4
```

You can also create the `data.frame` directly, i.e., without creating the `vectors` first:

```
#Making a data.frame directly
#You can put the two vectors on the same line too, but it is nicer to read this way.
data = data.frame(x = c(5,6,7,8),
                  y = c(1,2,3,4))
print(data)

##   x y
## 1 5 1
## 2 6 2
## 3 7 3
## 4 8 4
```

But, note the different `vectors` of the `data.frame` *have* to be of the same length, otherwise you will get an error. Try the following:

```
#Making a data.frame
#Two different types of vectors in the same data.frame
data = data.frame(x = c(5,6,7),
                  y = c(1,2,3,4))

## Error in `data.frame()`:
## ! arguments imply differing number of rows: 3, 4
```

You can create a `data.frame` with diffent types of `vectors`, as long as all the elements within a `vector` are the same type of element:

```
#Making a data.frame directly
data2 = data.frame(x = c("a","b","c","d"),
                   y = c(1,2,3,4))
print(data2)

##   x y
## 1 a 1
## 2 b 2
## 3 c 3
## 4 d 4
```

4 Subsetting with data types

By "subsetting", I mean getting a part of the data from that data type.

4.1 Subsetting from a `vector`

For `vectors`, you can mention the position of the element or a range of elements within square brackets to identify the corresponding element.

```r
#Making a vector
x = c("a","b","c","d")

#For the 3rd element
x[3]

## [1] "c"

#For all the elements from 3rd-4th positions
#Note, the colon allows you to get a sequence of numbers
x[3:4]

## [1] "c" "d"

#For the elements from a random list of positions
x[c(1,3,4)]

## [1] "a" "c" "d"
```

4.2 Subsetting from a `data.frame`

For a `data.frame`, there are a few ways in which you can subset. I will teach you one way that I think will be useful in the long run. Use the "$" to isolate the column/`vector` in the `data.frame`, and then use the bracket notation to isolate the values of interest.

```r
#Making a data.frame
data3 = data.frame(x = c("a","b","c","d","e","f"),
                   y = c(6:11))

#For the 3rd element from the "x" column
data3$x[3]

## [1] "c"

#For all the elements from 3rd-4th positions from the "x" column
data3$x[3:4]

## [1] "c" "d"

#For the elements from a list of positions from the "x" column
data3$x[c(1,3,4)]

## [1] "a" "c" "d"
```

What if you didn't know the name of the column, but knew the column number?

Advice: Though it is logically possible that you don't know the name of some column, and I show you how to access information in such a case. Honestly speaking, if you don't know the column names, then you should look at your data more carefully. There is no point trying to do data analysis if you don't know what you are analysing. For this reason the "$" notation is going to be almost always the preferred way of referring to columns.

```r
#If you don't know the column name
#For the 4th element from the 1st column
data3[4,1]
```

```
## [1] "d"
```

5 Useful programming elements

5.1 Conditionals or `if…else` statements

These are useful if you want to do something only if something else is TRUE/FALSE. For example, print "Hello World", if the 3rd value of a vector is equal to 6.

```
#Creating vector
x = c(4,7,6,1,2,10)

#Basic conditional
#For conditional "equal to", there need to be two "=".
if(x[3] == 6){
  print("Hello World")
}

## [1] "Hello World"

#For "greater than or equal to"
if(x[3] >= 6){
  print("Hello World")
}

## [1] "Hello World"

#For "greater than"
if(x[3] > 6){
  print("Hello World")
}

#For "less than or equal to"
if(x[3] <= 6){
  print("Hello World")
}

## [1] "Hello World"

#For "less than or equal to"
if(x[3] < 6){
  print("Hello World")
}
```

Conditionals can also have an "else" part that is done if the conditional is FALSE.

```
#Creating vector
x = c(4,7,6,1,2,10)

#Basic conditional
# For "equal to", there need to be two "=".
if(x[3] == 7){
  print("Hello, World!")
}else{
```

```
  print("Hi, fool!")
}

## [1] "Hi, fool!"
```

5.2 Loops or `for` statements

These are useful if you want to do something multiple times. There are many types of loops in R (`for` loops, `while` loops, `repeat` loops, …), but we will only work with `for` loops as these are likely to work for most looping issues you might face. `for` loops require a counter of some sort within them, as they execute a set of commands a prespecified number of times. Below, I use the counter or variable "i", but you could use any name/word that you think is clear.

```
#Basic for loop
for(i in c(1:6)){
  print("Hello")
}

## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"

#Using "if statements" within "for loops" affords a lot more power

#Create vector
x = c(1:6)

#Looping thru each value and checking if the value matches "4"
for(i in x){
  if(i == 4){
    print("Yippie!")
  }else{
    print(":(")
  }
}

## [1] ":("
## [1] ":("
## [1] ":("
## [1] "Yippie!"
## [1] ":("
## [1] ":("
```

6 Some useful references

A lot of doubts/questions you will have are very likely already answered online. Search for whatever your doubt is and include "Stack Overflow" or "Cross Validated" to your search string on Google. It will most probably show you links to the solutions to the problem/doubt on the online community **Stack Overflow** or **Cross Validated**, which are both fantastic online communities for programming or statistics related info, respectively.

7 Homework

1. **Find the error in each of the following pieces of code**:

   (a) When trying to print a string, and you write the following:

   ```
   print(Hello World!)
   ```

   (b) What is the output of the following code? This is not quite an 'error', but explain the output and why that was the output.

   ```
   x = c(5,6,7)
   y <- c(1,2,3,4,5)
   x+y
   ```

   Note: You can use '<-' or '=' as an assignment operator. Try it for yourself — both work. The former is largely due to historical reasons having to do with keyboard layouts. I will use the second way throughout this course. But, this comes down to individual preference. So, feel free to use the other one if you are more used to it. One important thing: be consistent about your choice so that you don't confuse yourself.
   For most purposes, both work the same way. You can read about some subtle difference about the scope of operation amongst other things on Stack Overflow.

   (c) When identifying a particular value in a vector, and you write the following:

   ```
   values = c('Happy',"Sad","Apathetic","Hyper")
   values(3)
   ```

   Note: While you can use either single quotes or double quotes to assign character/string values in vector, I would recommend being consistent.

   (d) When calculating the mean, and you write the following:[1]

   ```
   values = c(1, 2, 3, 5  6)
   mean(values)
   ```

   (e) When creating a `data.frame`, and you write the following

   ```
   randomData = data.frame(x = c(5,6,7),
                           y = c(1,2,3,4,5))
   ```

   (f) When your write the following conditional statement to print the number 10 if x = 5.

   ```
   x=5
   if(x = 5){
     print(10)
   ```

   (g) When trying to loop through the same instructions multiple times

   ```
   for(2025_counter in c(1:25)){
     print(10)
   }
   ```

2. Write a program to do the following in the order listed. Make sure to comment your code well.

   A First create a `data.frame` named **experiment** with 3 columns (names = *Subj, Group, Reaction-Time*)

[1] This exercise was inspired by a similar exercise in Bodo Winter's textbook.

B There should be 10 values in each column.

i. Column 1: the numbers 101-110.

ii. Column 2: the letters "A" and "B" alternated.

iii. Column 3: any 10 random non-integer numbers.

C Assign whatever is the 6^{th} value of the last column (*ReactionTime*) of the `data.frame` to a new variable named **answer**. [Note, you need to use the `data.frame` **experiment** and the column name (*ReactionTime*) in your code for this task, and you need to be able to write the code so that it prints out the right value even if you didn't know what was in that particular position of the `data.frame`.]

(a) Print the value in the variable **answer**.