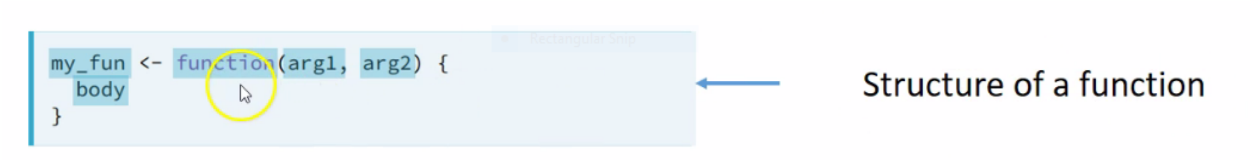
Now we will be learning, how to write our own function? Before learning this, we have to learn where we actually create a function. Whenever we are doing a lot of copy paste of the same code with minimal change we just create our own function. Whenever you are copy pasting lot of time the same number of code you are copy pasting again and again then it’s a time to create your own function.

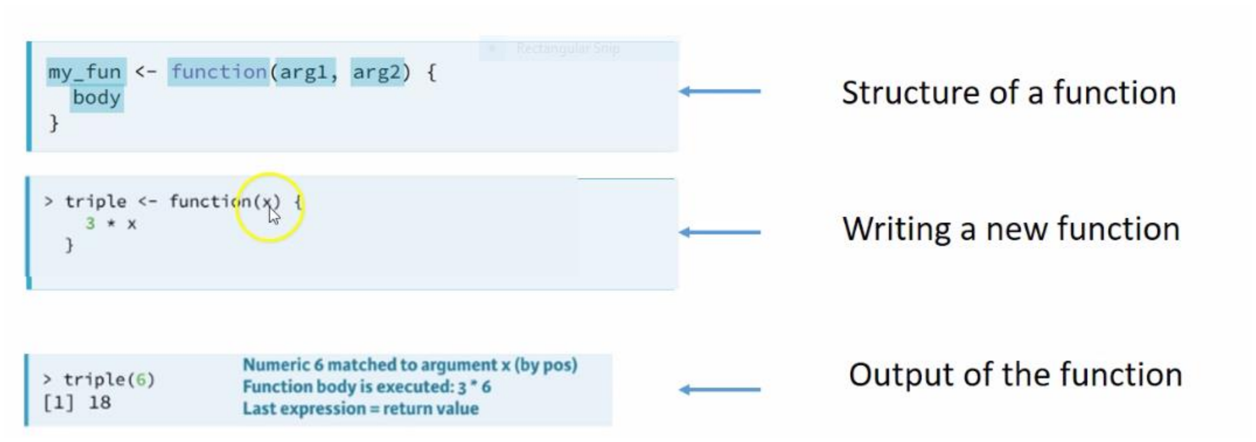
Why would actually create a function? Writing your own functions is one way to reduce duplication in your code. It is easier to use. Not Only easier to use but also easier to understand and its actually removing lots of your deduplication work so that’s why we create a function.

Structure Function(): Structure is as you name your function, like mean, standard deviation, range or append etc some kind of meaningful name you have to do.



```
my_fun <- function(arg1, arg2) {
  body
}
```

Suppose, you want to create a function and the function will give you anything which it passed you, and then you will give it triple. That so, you will multiply it that 3 and you will return the value. So here, the meaningful name is triple and how to write that, that is our query. Perhaps, you start with the name > triple <- then you are writing the function and then write down how many arguments you have? Here, suppose you have only one argument that’s why you write ‘x’ and then you have to create that body in the function and whatever you take for your input that is ‘x’ and then multiply it with 3 and then return it. Now the function is being created. (>triple<- function(x) {3*x} and now if you pass 6 in your triple function you will get 18. Here we are working with one argument which is pretty easy,next we are doing with two argument.



```
my_fun <- function(arg1, arg2) {
  body
}
```

Structure of a function

```
> triple <- function(x) {
    3 * x
  }
```

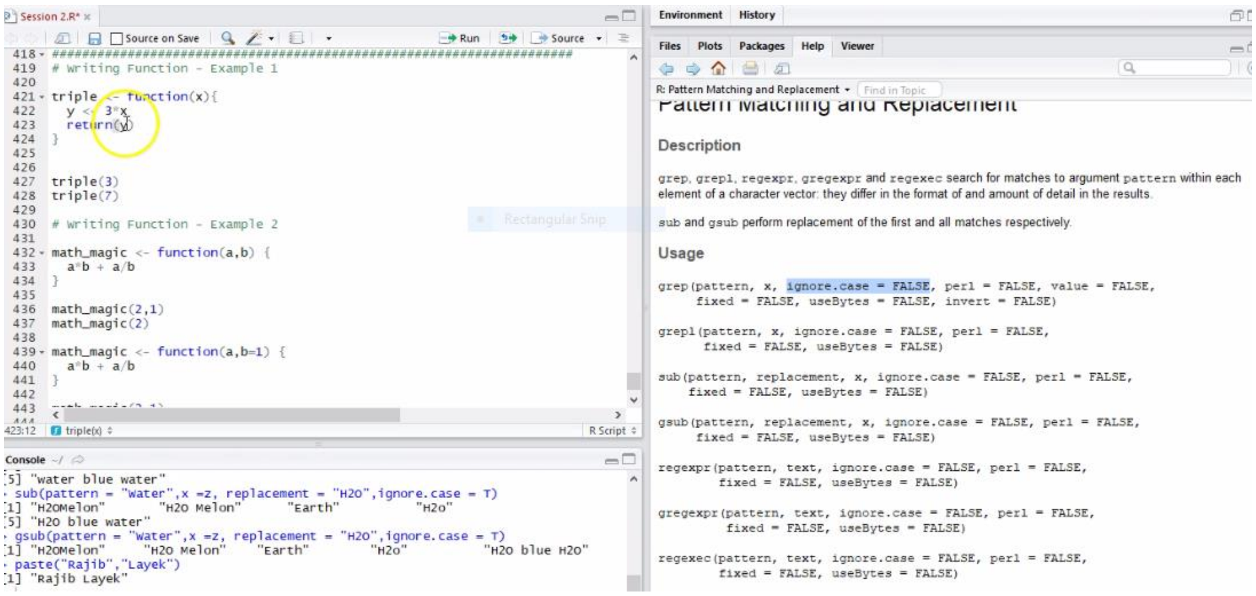
Writing a new function

```
> triple(6)
[1] 18
```

Numeric 6 matched to argument x (by pos)
Function body is executed: 3 * 6
Last expression = return value

Output of the function

In the second argument we will be creating a new function called math magic which takes actually two arguments suppose one is ‘a’ and another is ‘b’,and you pass ‘a’ multiply ‘b’ plus ‘a’ by ‘b’ like “a*b+a/b” that’s why its work suppose this the formula and we have created math magic. If you pass a=2 and b=1 and you are getting 4 as a result. Here,2 multiple 1 equal to 2 and 2 by 1 equal to 2 and then 2 plus 2 equal to 4, so the result is 4 as well. (2*1+2/1=4)



```
#########################################################################
# writing Function - Example 1

triple <- function(x){
  y <- 3*x
  return(y)
}


triple(3)
triple(7)

# writing Function - Example 2

math_magic <- function(a,b) {
  a*b + a/b
}

math_magic(2,1)
math_magic(2)

math_magic <- function(a,b=1) {
  a*b + a/b
}
```

```
[5] "water blue water"
> sub(pattern = "Water",x =z, replacement = "H2O",ignore.case = T)
[1] "H2OMelon"         "H2O Melon"        "Earth"            "H2o"
[5] "H2O blue water"
> gsub(pattern = "Water",x =z, replacement = "H2O",ignore.case = T)
[1] "H2OMelon"       "H2O Melon"      "Earth"          "H2o"            "H2O blue H2O"
> paste("Rajib","Layek")
[1] "Rajib Layek"
```

Pattern Matching and Replacement

Description

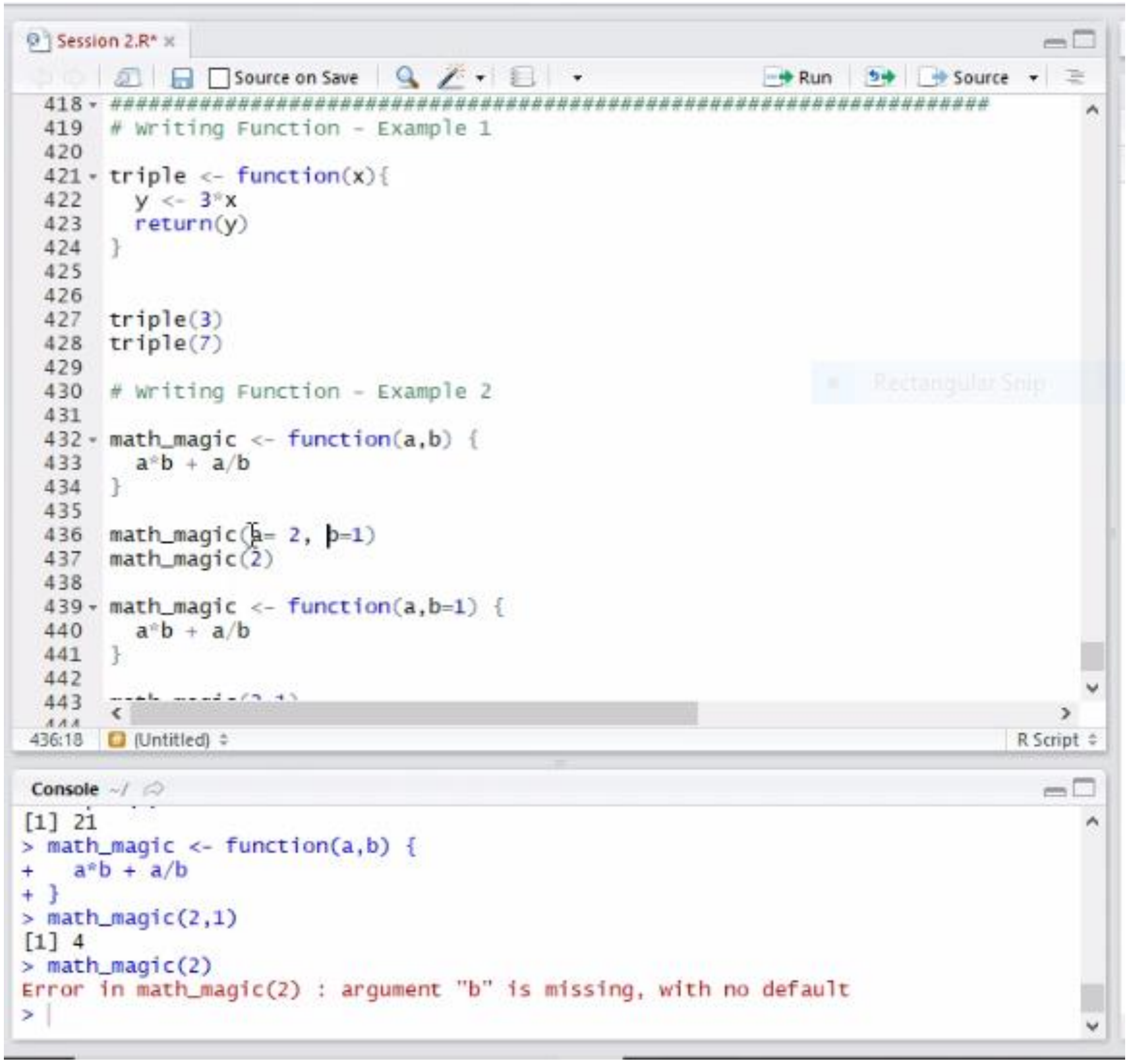
grep, grepl, regexpr, gregexpr and regexec search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

sub and gsub perform replacement of the first and all matches respectively.

Usage

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
     fixed = FALSE, useBytes = FALSE, invert = FALSE)

grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
      fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
    fixed = FALSE, useBytes = FALSE)

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
     fixed = FALSE, useBytes = FALSE)

regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
         fixed = FALSE, useBytes = FALSE)

regexec(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)
```

Somehow you don’t remember the second value which is for 'b' and then you get an error. Remember instead of this you could be written it this way also. like, you write ‘a’ equal to 2 and ‘b’ equal to 1. You can do this By name and since the position also same, that’s a reason the function is working. But if you forget to give the second value then you get an error. That’s why, whenever you are creating your function try to give a meaningful default value as well.



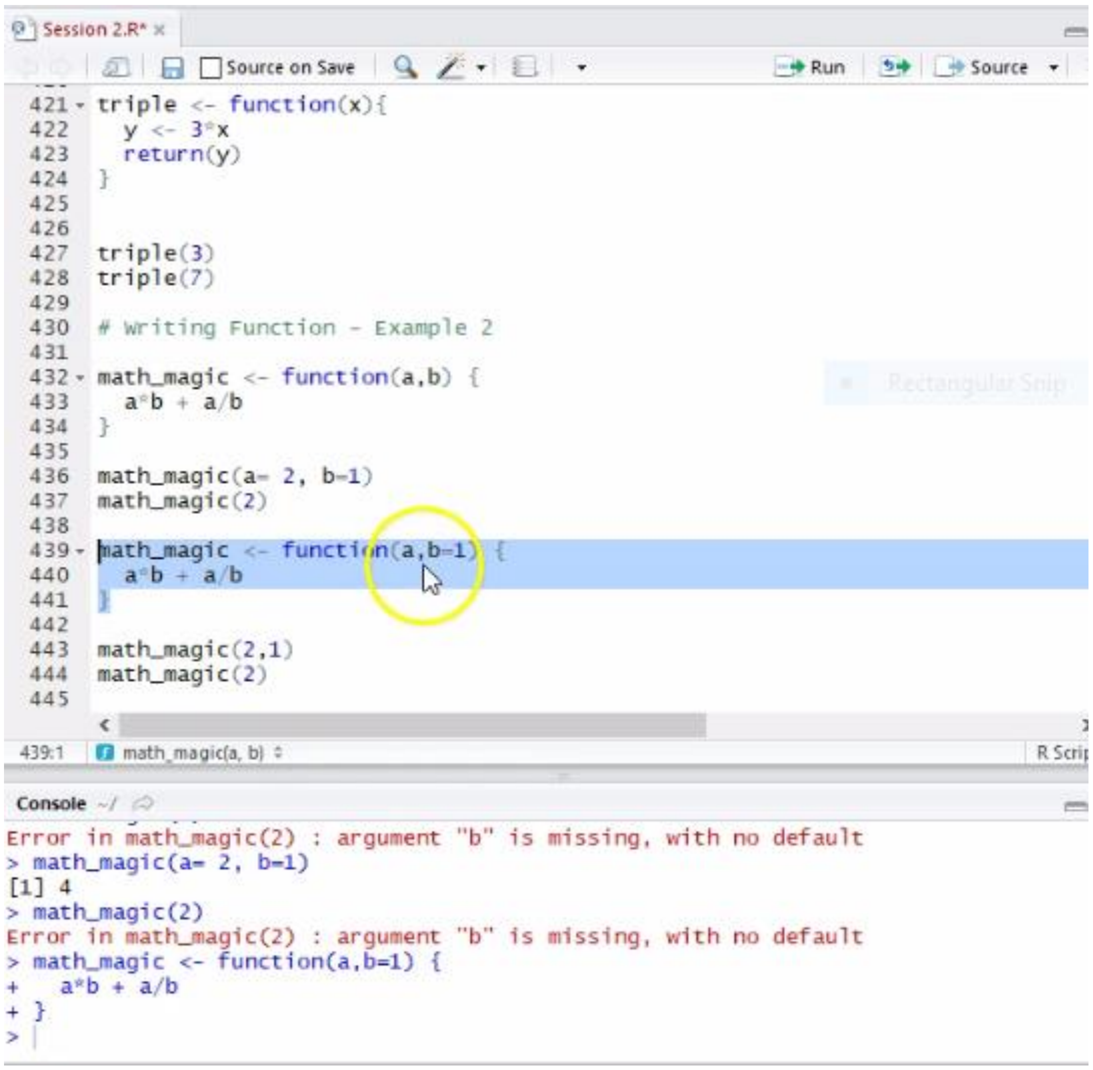
Session 2.R*

```
###########################################################################
# Writing Function - Example 1

triple <- function(x){
  y <- 3*x
  return(y)
}


triple(3)
triple(7)

# Writing Function - Example 2

math_magic <- function(a,b) {
  a*b + a/b
}

math_magic(a= 2, b=1)
math_magic(2)

math_magic <- function(a,b=1) {
  a*b + a/b
}

```

Console ~/

```
[1] 21
> math_magic <- function(a,b) {
+   a*b + a/b
+ }
> math_magic(2,1)
[1] 4
> math_magic(2)
Error in math_magic(2) : argument "b" is missing, with no default
>
```

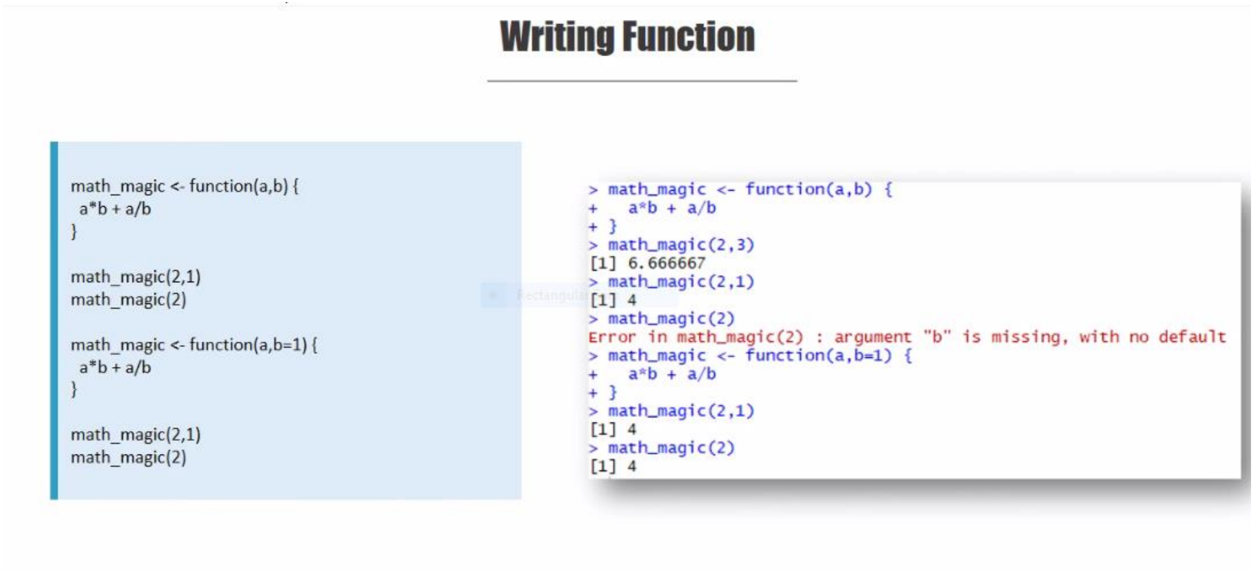
Here we are passing a default value ‘b’ equal to 1 and get the same result and same set of code but we have set the default value that ‘b’ would be 1, if we pass 2 and 1 then the result would be same, here if we forget to give the value of ‘b’ still we get a result. So earlier when it gave, we are getting an error result but now it automatically takes the default value as 1 that’s why we get the result.



```
triple <- function(x){
  y <- 3*x
  return(y)
}


triple(3)
triple(7)

# writing Function - Example 2

math_magic <- function(a,b) {
  a*b + a/b
}

math_magic(a= 2, b=1)
math_magic(2)

math_magic <- function(a,b=1) {
  a*b + a/b
}

math_magic(2,1)
math_magic(2)
```

```
Error in math_magic(2) : argument "b" is missing, with no default
> math_magic(a= 2, b=1)
[1] 4
> math_magic(2)
Error in math_magic(2) : argument "b" is missing, with no default
> math_magic <- function(a,b=1) {
+   a*b + a/b
+ }
> 
```

That is the way, how to create your own function. It is a simple way. And above example also very simple. But whenever you are doing a lot of data manipulation where has a lot of columns and lot of attributes and you want to do a similar kind of operation on each of those attributes then instead of doing and writing again and again you are feel bore so what can you do? you can simply create a function which can look after each of those columns one by one and that’s why you can reuse the number of code.



Writing Function

```
math_magic <- function(a,b) {
  a*b + a/b
}

math_magic(2,1)
math_magic(2)

math_magic <- function(a,b=1) {
  a*b + a/b
}

math_magic(2,1)
math_magic(2)
```

```
> math_magic <- function(a,b) {
+    a*b + a/b
+ }
> math_magic(2,3)
[1] 6.666667
> math_magic(2,1)
[1] 4
> math_magic(2)
Error in math_magic(2) : argument "b" is missing, with no default
> math_magic <- function(a,b=1) {
+    a*b + a/b
+ }
> math_magic(2,1)
[1] 4
> math_magic(2)
[1] 4
```

This brings an end to this post, I encourage you to re read the post to understand it completely if you haven’t and THANK YOU.