R notes

Bharvi Dhall

RStudio gives you a way to talk to your computer. R gives you a language to speak in. Credits: Garrett Grolemund, Hadley Wickham - Hands-On Programming with R_ Write Your Own Functions and Simulations-O'Reilly Media (2015)

##The Basics of R

```
1:100
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
# [1] represents first line , second line starts with 25, third line with 49.
```

To store information in R we use an object.

```
a <- 1:10  #<- is the assignment operator
a  #displays information in a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
b <- c('a','b','c')  #The character values are entered using ''
b
```

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

```
#to extract all objects
ls()
```

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

##EXAMPLE

#Lets create a virtual die and name it die

```
die <- 1:6

# We can do any operations on this

die +1  # this adds one to all elements in the die
```

```
## [1] 2 3 4 5 6 7
```

```
die * die
```

```
## [1]  1  4  9 16 25 36
```

Note: When we multiply die * die does not always follow the rules of matrix multiplication. Instead, R uses element-wise execution.

If you give R two vectors of unequal lengths, R will repeat the shorter vector until it is as long as the longer vector, and then do the math. This isn't a permanent change—the shorter vector will be its original size after

R does the math. If the length of the short vector does not divide evenly into the length of the long vector, R will return a warning message. This behavior is known as vector recycling, and it helps R do element-wise operations:

```
1:4
```

```
## [1] 1 2 3 4
```

```
die + 1:4
```

```
## Warning in die + 1:4: longer object length is not a multiple of shorter
## object length
```

```
## [1] 2 4 6 8 6 8
```

```
# here the first element of die added to first element of vector, the second element of die to second e
```

R also performs traditional multiplication.It can do inner multiplication with the %*% operator and outer multiplication with the %o% operator

```
die %*% die
```

```
##      [,1]
## [1,]   91
```

```
# 1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6
```

```
die %o% die
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    4    5    6
## [2,]    2    4    6    8   10   12
## [3,]    3    6    9   12   15   18
## [4,]    4    8   12   16   20   24
## [5,]    5   10   15   20   25   30
## [6,]    6   12   18   24   30   36
```

```
# normal matrix multiplication
```

Some inbuilt Functions

```
round(2.4566)  # rounds up to 2
```

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

```
round(3.5555) # rounds up to 4
```

```
## [1] 4
```

```
mean(1:6)  #mean of 1,2,3,4,5,6 is 3.5
```

```
## [1] 3.5
```

```
#There are alot more functions that you can see in your own time
```

If we want to sample randomly from a bunch , R can do it with its inbuilt function sample

```
sample(1:100, size=4)   #this takes a sample of four elements from 100 numbers.
```

```
## [1] 17 27 34 20
```

```r
# if you are unsure of the arguments in inbuilt function, use args()

args(sample) #by default for this function, replace is false.
```

```
## function (x, size, replace = FALSE, prob = NULL)
## NULL
```

#Writing your own functions

The functions are made up of three parts: a body of code, and a set of arguments. To make your own function, you need to replicate these parts and store them in an R object, which you can do with the function function.

Syntax my_function <- function() {}

Now lets write a function, to roll a die twice, get 2 values and find its sum.

```r
roll <- function(){
  die <- 1:6
  s<- sample(die, size = 2, replace = TRUE)     #to create independent random samples, we use replace =
  sum(s)
}

roll()   #it will give you different values each time
```

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

The above function just works for the values of die, if you want to make the function more generic. We can pass the name of the data from which we have to sample as an argument in the function.

The arguments in the functions can be taken as an input from the user.

```r
roll_a <- function(new){
  die <- 1:6
  s<- sample(new, size = 2, replace = TRUE)     #to create independent random samples, we use replace =
  sum(s)
}

roll_a(new = 1:80)
```

```
## [1] 130
```