R Notebook Introduction to R in Immunobiology course

This is the R file I used to go through the introduction to R. I hope it will help you to understand things better. We will also use the presenters wall during this lecture at: immbio.presenterswall.nl

Variables in R: Numeric:

```
a <- 5
print (a)
```

```
## [1] 5
```

Strings/characters:

```
?nchar
x<-"my first lecture in R"
nchar(x)
```

```
## [1] 21
```

Mathematical operations: R is a powerful calculator. You can use many operators in R (see Reference guide). ^ operator takes the power of the numbers.

```
d <- 5
d <- d ^ d
print (d)
```

```
## [1] 3125
```

Vectors:

Everything is a vector, or a high dimensional vector in R. c() function makes vectors. length() function prints the length of a vector. The vectors can contain any type of variables, numbers, characters, etc.

```
e <- c(10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
print (length (e))
```

```
## [1] 11
```

```
#But this is not so nice code, you can also do this.
e<- c(10:0)
print(length(e))
```

```
## [1] 11
```

```
#You can even take out c()
e<- 10:0
print(length(e))
```

```
## [1] 11
```

```
x <- c("my", "first", "lecture", "in", "R")
nchar(x)
```

```
## [1] 2 5 7 2 1
```

```
mystrings <- c("I", "love", "R")
print (mystrings)
```

```
## [1] "I"    "love" "R"
```

```
print ( length(mystrings))
```

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

Logical values (TRUE or FALSE) is very important in R. You can use them to select the data you want to have. Here we will look into few operators that return logical values: - ">" and "<" larger - "==" is equal? Take care: it is different than "=" - "!=" not equal - is.numeric, is.character

```
h <- 0
phorty_phour <- 44
h > phorty_phour
```

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

```
h< phorty_phour
```

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

```
h == phorty_phour
```

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

```
h != phorty_phour
```

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

```
is.numeric(h)
```

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

```
is.character(mystrings)
```

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

```
is.numeric(mystrings)
```

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

```
mystrings
```

```
## [1] "I"    "love" "R"
```

```
mystrings[c(TRUE, FALSE, FALSE)]
```

```
## [1] "I"
```

Flow control in R:

Let's learn about "if and else" command.

```
b <- 1.52 - 0.02
c <- 1 + 0.41
if (b <= c) {
print (b) } else {
  print ("Nothing happens")}
```

```
## [1] "Nothing happens"
```

For loops are difficult to understand at first. But once you understand them, you will see that they are very powerfull. Let's go step by step with this code.

```
f <- 0
for (kk in 1:5) { # kk is our counter, it can have any name, xx, count, etc.
f <- f + kk
```

```
output<- sprintf ("kk is %d, f is %d",kk, f)
print(output)}
```

```
## [1] "kk is 1, f is 1"
## [1] "kk is 2, f is 3"
## [1] "kk is 3, f is 6"
## [1] "kk is 4, f is 10"
## [1] "kk is 5, f is 15"
```

You can use the logical values in if() functions to select what you want. If() functions can be embedded in anywhere in your code: in a for loop, in a function etc. We have already used if() function above.

```
i <- 0:6
i
```

```
## [1] 0 1 2 3 4 5 6
```

```
i[4]
```

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

```
for (j in i) {
if (j != i[4]) {
print (j) } }
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 4
## [1] 5
## [1] 6
```

Let us make one example in presenters wall to get this right.

```
l <- 6:2
for (k in l) {
if (k > l[3]) {
print (k) } }
```

```
## [1] 6
## [1] 5
```

While loops is another way of controlling your code. R will go on performing the codes until the condition in while is no longer true. In other words

while (TRUE) { do something } if FALSE don't do anything

```
k<-1
while (k<4){
  cat("Hello world\n")
  k<-k+1
}
```

```
## Hello world
## Hello world
## Hello world
```

```
k <- 7
while (k>0) {
```

```
print (k)
k <- k-1 }
```

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

```
#Remember for R, FALSE = 0, and TRUE = 1

k <- 7
while (k) {
print (k)
k <- k-1 }
```

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

Let's now make an example in presenters wall.

```
days_left <- 5
while (days_left > 0) {
  cat ("Still no vacation\n")
  days_left<- days_left-1
  }
```

```
## Still no vacation
## Still no vacation
## Still no vacation
## Still no vacation
## Still no vacation
```

You can also use for loops with characters

```
m <- c("check1", "check2", "check3", "check4")
for (i in 1:4) {
print (m[i]) }
```

```
## [1] "check1"
## [1] "check2"
## [1] "check3"
## [1] "check4"
```

paste() function glues variables/strings etc.

```
d <- "concate"
at <- "nation"
b<- 3
paste (d, at)
```

```
## [1] "concate nation"
```

```r
paste (d, b)
```

```
## [1] "concate 3"
```

```r
# If you don't want to have a space
paste (d, at, sep="")
```

```
## [1] "concatenation"
```

Functions:

If it is required that the same operation is performed many times, we often use functions, that are like little algorithms that perform a simple operation. Most computer languages know hundreds of functions that can do something, such as convert your input data into a different format, go through a list of values one by one, generate a plot, etcetera. In R you can write your own functions, and during the computer exercises you will use already made functions to perform mathematical analysis.

```r
#Let's start easy
firstfun <- function(name) {
  cat("Hello ")
  cat(name)
}
firstfun("Bas")
```

```
## Hello Bas
```

Write a function in R that you can give any DNA sequence as input, and that will return the length of the sequence.

```r
firstfun <- function(seq) {
  cat(seq)
  cat("\n")
  nchar(seq)
}
firstfun("acgtggggacac")
```

```
## acgtggggacac
```

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

```r
firstfun("Can")
```

```
## Can
```

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

```r
firstfun <- function(seq) {
  cat(seq)
  cat("\n")
  nchar(seq)
}
```

Write a function that prints out your name and your student number. Call the function to see that it works as you want.

Now a more complicated example:

```r
whoami <- function(name, number){
cat("studentname:", name, ", studentnumber:", number)
}
```

```
# Usage:
whoami("Can Kesmir", "007")
```

```
## studentname: Can Kesmir , studentnumber: 007
```

```
euclid <- function (values) {
A <- values[1]
B <- values[2]
while (B != 0) {
if (A > B) {
A <- A - B }
else {
B <- B - A }
}
return (A)
}
```

Let's see what do you think euclid does in presenters wall.

How would you call this function from within an R script, to obtain the largest common denominator of two numbers 43 and 292 in a variable called lcd? Try this on your laptop.

```
# euclid(43, 292) This won't work, because we need to give the input as a vector
euclid(c(44, 231))
```

```
## [1] 11
```

```
# I could also have done:
myvec <- c(44, 231)
euclid(myvec)
```

```
## [1] 11
```

```
# Let us save the out put in a variable called lcd
lcd <- euclid(myvec)
lcd
```

```
## [1] 11
```

What would happen if you called the function euclid with three parameters, 43, 292, and 11?

```
euclid(c(42, 36, 63))
```

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

```
euclid(c(42, 63, 36))
```

```
## [1] 21
```

```
euclid(c(42, 63)) # this function makes use of only the first two values in a vector.
```

```
## [1] 21
```

```
                  # the third value is not used in the calculations.
euclid(c(43, 292, 11))
```

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

Let's make a presenters wall question:

```
decom  <- function (x) {
    as.numeric(gsub(",", "", x))
```

```
}
decom("32,900,888")
```

```
## [1] 32900888
```

Write a script in R that rounds a decimal number to the nearest integer.

```
round<-function(value){
  if (as.integer(value+0.5) >= as.integer(value))
      as.integer(value+0.5)
  else as.integer(value)
}
x<-3.51
round(x)
```

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

Reading data: Download the tab-delimited text file in https://tbb.bio.uu.nl/immbio/R/fig4.tsv.

a. How many rows and columns are in the table? Check your answer in R.

```
cohort <- read.table("fig4.tsv", header=TRUE, sep="\t", stringsAsFactors = FALSE)
dim(cohort) # the first number dim gives is the rows and the second number is the                #column
```

```
## [1] 6 7
```

c-d. Use an R command to discover the data type of cohort. Because we indicated that HEADER=TRUE, each column now has a name that we can use to access its values. Type cohort$Age to see all the values in the Age column.

```
str(cohort)
```

```
## 'data.frame':    6 obs. of  7 variables:
##  $ Name      : chr  "Patient01" "Patient02" "Patient03" "Patient04" ...
##  $ First_Name: chr  "Adriana" "Matthias" "Jan" "Amin" ...
##  $ Last_Name : chr  "Mattos" "Uyttenhaghe" "Aerntsz" "Abboud" ...
##  $ Age       : int  35 56 34 67 31 73
##  $ Weight    : num  64.5 78.5 50.3 89.6 87.5 69.4
##  $ Gender    : chr  "F" "M" "M" "M" ...
##  $ Married   : logi  TRUE FALSE TRUE TRUE FALSE TRUE
```

```
#If you want to easily access patients
cohort <- read.table("fig4.tsv", header=TRUE, sep="\t", stringsAsFactors = FALSE, row.names = 1)
str(cohort)
```

```
## 'data.frame':    6 obs. of  6 variables:
##  $ First_Name: chr  "Adriana" "Matthias" "Jan" "Amin" ...
##  $ Last_Name : chr  "Mattos" "Uyttenhaghe" "Aerntsz" "Abboud" ...
##  $ Age       : int  35 56 34 67 31 73
##  $ Weight    : num  64.5 78.5 50.3 89.6 87.5 69.4
##  $ Gender    : chr  "F" "M" "M" "M" ...
##  $ Married   : logi  TRUE FALSE TRUE TRUE FALSE TRUE
```

```
cohort["Patient01",]
```

```
##           First_Name Last_Name Age Weight Gender Married
## Patient01    Adriana    Mattos  35   64.5      F    TRUE
```

```
cohort$Age
```

```
## [1] 35 56 34 67 31 73
```

```
cohort$Weight
```

```
## [1] 64.5 78.5 50.3 89.6 87.5 69.4
```

Use one R command to calculate the average of all the values in the Weight column.

```
?mean
mean(cohort$Weight)
```

```
## [1] 73.3
```

```
mean(cohort[,"Weight"])
```

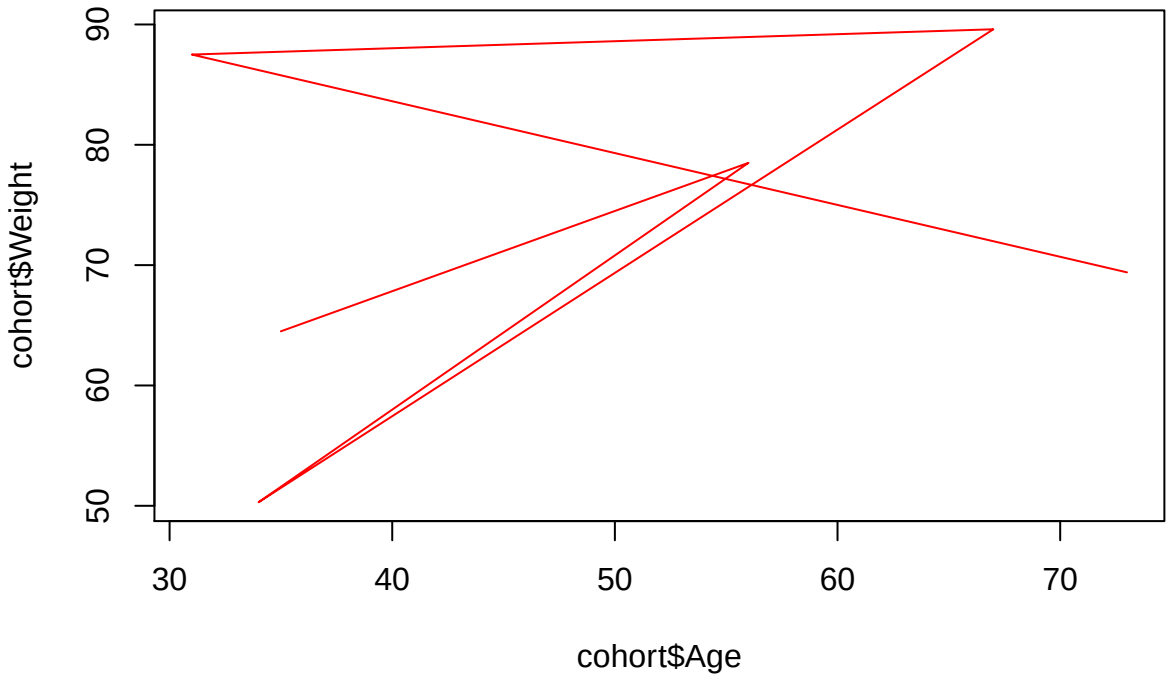
```
## [1] 73.3
```

```
summary(cohort$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   31.00   34.25   45.50   49.33   64.25   73.00
```

Plot a scatter plot of the age versus the weight of the individuals in Figure 4 using a red line.

```
plot(cohort$Age, cohort$Weight, type="l", col="red")
```



b. Why does your plot look so messy?

```
cohort
```

```
##           First_Name   Last_Name Age Weight Gender Married
## Patient01    Adriana      Mattos  35   64.5      F    TRUE
## Patient02   Matthias Uyttenhaghe  56   78.5      M   FALSE
## Patient03        Jan     Aerntsz  34   50.3      M    TRUE
## Patient04       Amin      Abboud  67   89.6      M    TRUE
## Patient05      Janet  Thomlinson  31   87.5      F   FALSE
## Patient06 Frederique         Vos  73   69.4      F    TRUE
```

c. How could you make it look better? (Hint: use order().)

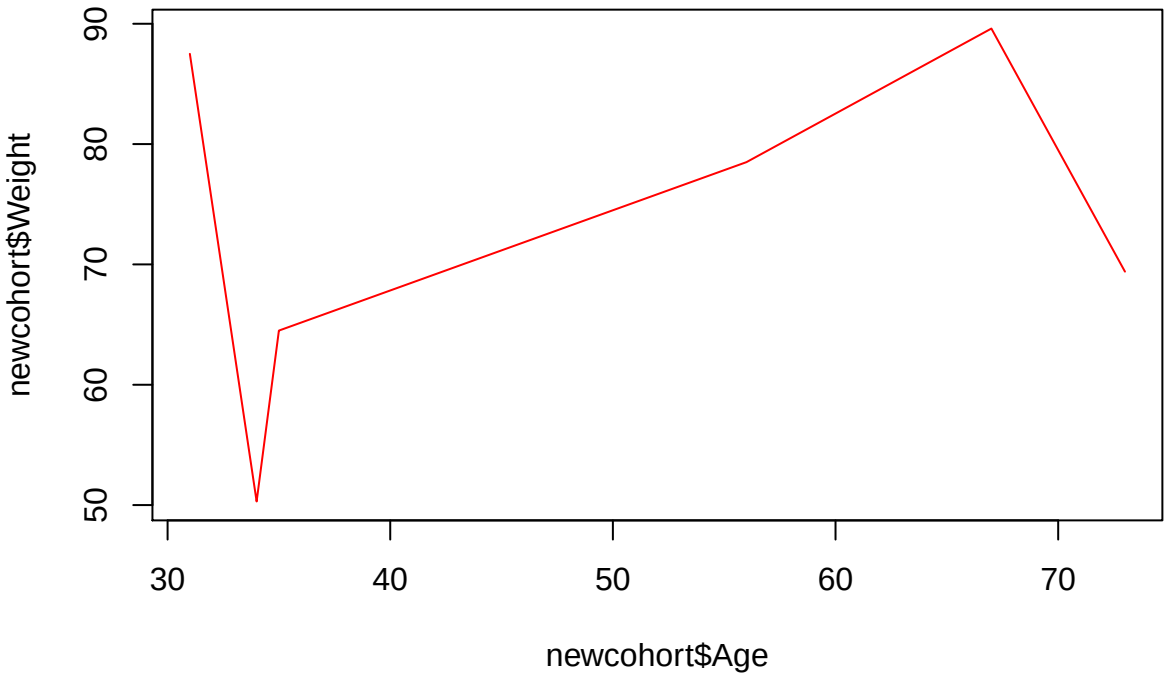
```
order(cohort$Age)
```

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

```
newcohort <- cohort[order(cohort$Age),]
newcohort
```

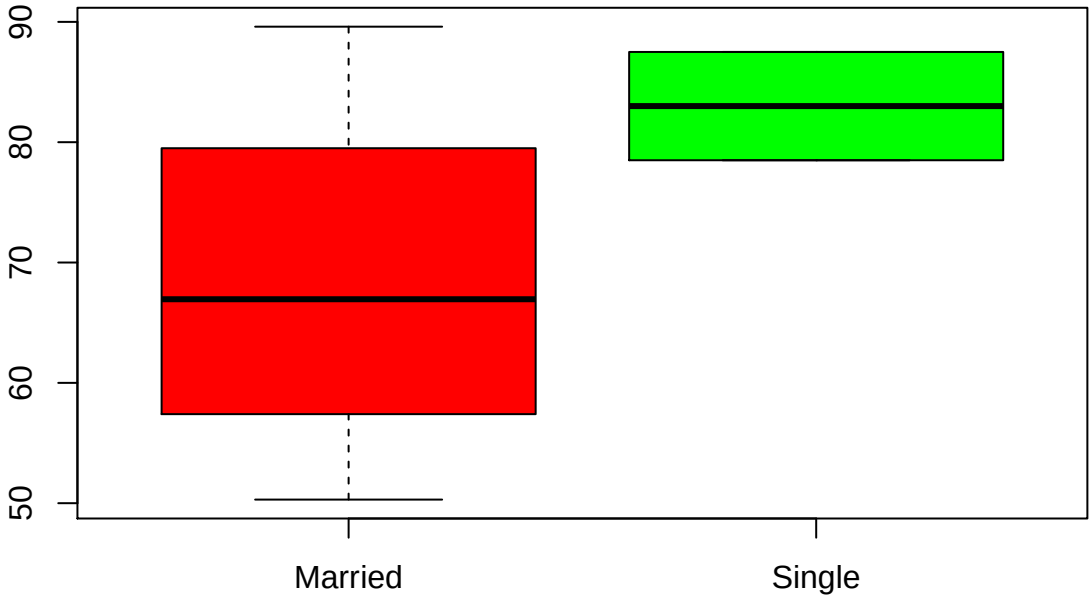
```
##           First_Name   Last_Name Age Weight Gender Married
## Patient05      Janet  Thomlinson  31   87.5      F   FALSE
## Patient03        Jan     Aerntsz  34   50.3      M    TRUE
## Patient01    Adriana      Mattos  35   64.5      F    TRUE
## Patient02   Matthias Uyttenhaghe  56   78.5      M   FALSE
## Patient04       Amin      Abboud  67   89.6      M    TRUE
## Patient06 Frederique         Vos  73   69.4      F    TRUE
```

```
plot(newcohort$Age, newcohort$Weight, type="l", col="red")
```



Make a boxplot to compare the weight of married and single people in the cohort.

```
boxplot(cohort[cohort$Married==TRUE,"Weight"], cohort[cohort$Married==FALSE,"Weight"], names=c("Married
```

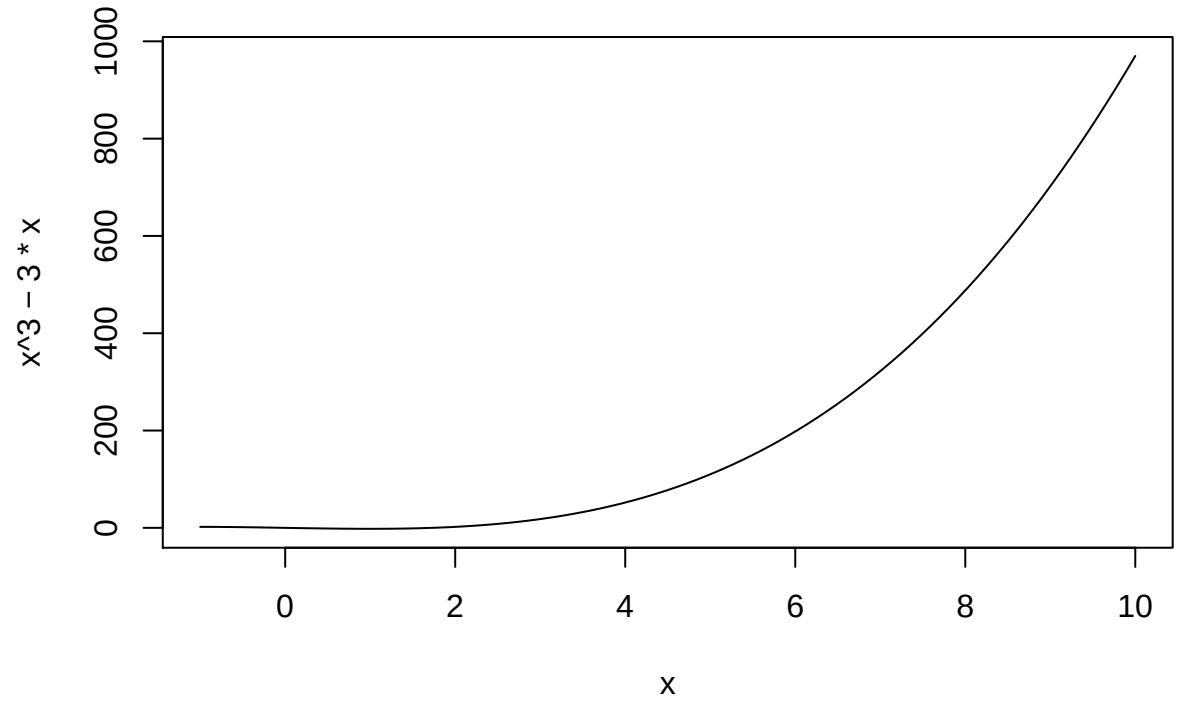



```
wilcox.test(cohort[cohort$Married==FALSE,"Weight"], cohort[cohort$Married==TRUE, "Weight"])
```

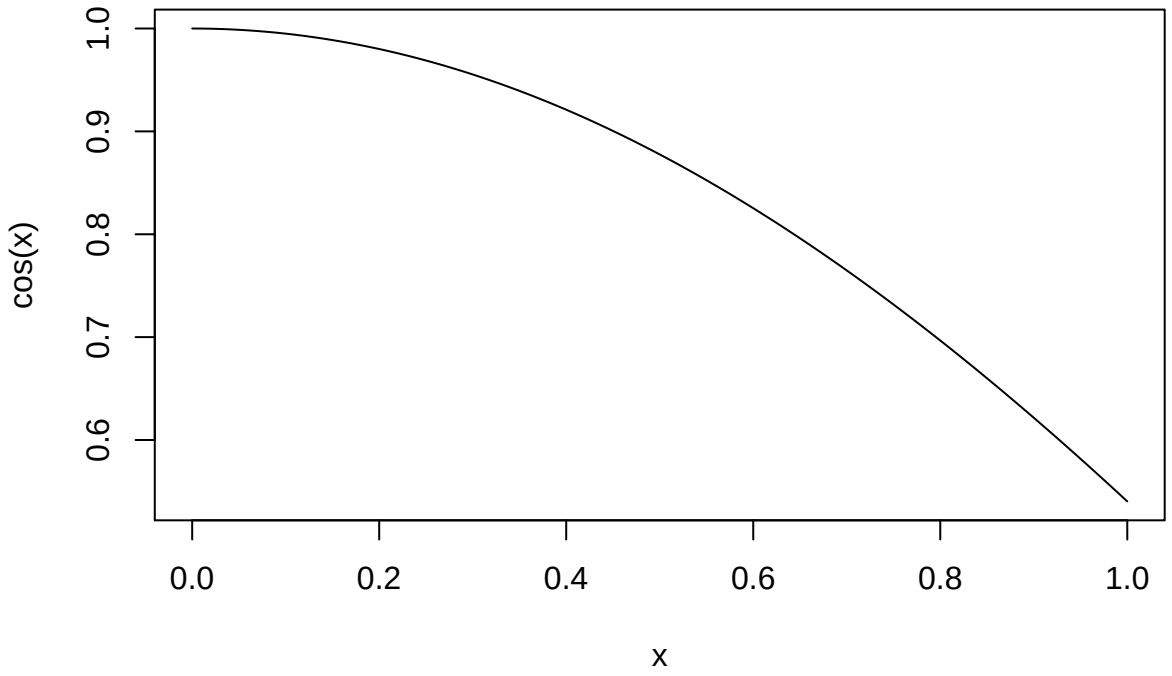
```
##
##  Wilcoxon rank sum test
##
## data:  cohort[cohort$Married == FALSE, "Weight"] and cohort[cohort$Married == TRUE, "Weight"]
## W = 6, p-value = 0.5333
## alternative hypothesis: true location shift is not equal to 0
```

What if you just want to plot a function???

```
curve(x^3-3*x, from =-1, to=10)
```



```
curve(cos)
```




```
myfunction<- function(p,s) {
r <- (1-p)^s
r
}
curve(myfunction(p=x,s=10), from = 0.01, to=0.99)
```