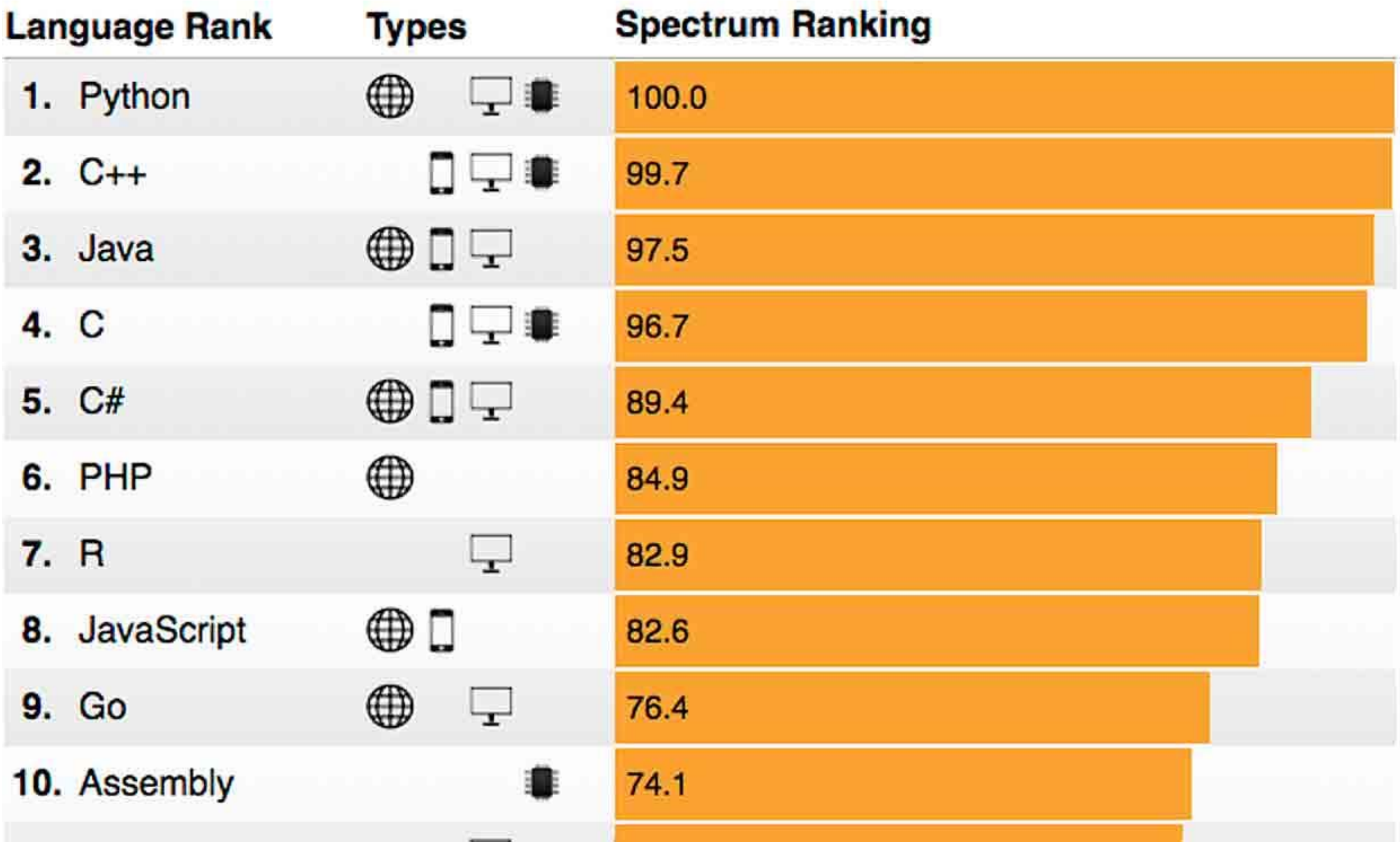
R 프로그래밍 #3

2019. 03. 20

한국생명공학연구원
김하성

Top-programming-languages (2018)



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ■	100.0
2. C++	📱 🖥 ■	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ■	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	■	74.1

https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages https://github.com/greendaygh/Rprog2019

In the last class

- Object – vector, factor, matrix, …
- vector – numeric, character, logical
- matrix – Two dimension numeric, char, logical..

```
> seq
> rep
> length
> which
> paste
> sample
> matrix
```

- matrix indexing

In the last class

- Define a function

```
my_sine <- function(x){
            y <- sin(x)
            return(y)
}
```

- Load once (Ctrl + Enter)
- Use

```
> my_sine(pi)
```

- This returns the sine of pi
  - one parameter: x
  - one argument: pi

Exercise 3-1) matrix

```
# make directory C:\Rprog\03
setwd(" C:/Rprog/03 ")
```

1. Build a matrix, ‘mymat’ consisting of 100 rows and 3 columns with initial value of 0
2. Generate 1 to 100 sequential values and save them to the first column of mymat
3. Generate 100 odd numbers ranging from 1 to 200 and save them to the second column of mymat
4. Generate 100 even numbers ranging from 1 to 200 and save them to the third column of mymat
5. Show values at 2, 3, 4, 5 rows in the second column
6. Substrate 1 from all elements of mymat
7. Substrate values in the second column from the values in the first column, and save them to ‘mysub’
8. Show the mean/sum/squared sum of mysub

Object – ~~vector~~, ~~factor~~, ~~matrix~~, data.frame, list

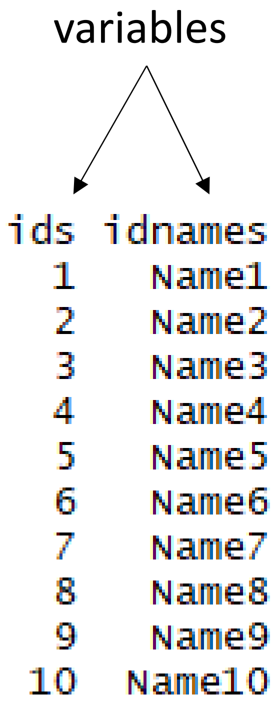
- Basic data structure in R
  - Numeric vector
  - Logical vector
  - Character vector
- Use ‘class’ function

```
> x <- c(10.4, 6.4, 1.7)
> class(x)
> x <- c(10.4, 6.4, 1.7, "test")
> matrix(c(1,2,3,4), 2, 2)
> matrix(c(1,2,3,"4"), 2, 2)
> matrix(c(1,2,3,TRUE), 2, 2)
```

Object - Data frames

- A preferred way to store data in R

```
ids <- 1:10
idnames <- paste("Name", ids, sep="")
students <- data.frame(ids, idnames)
students
class(ids)
class(idnames)
class(students)
class(students[,1])
class(students[,2])
students <- data.frame(ids, idnames, stringsAsFactors = F)
class(students[,2])
students[1,]
```



```
ids idnames
  1   Name1
  2   Name2
  3   Name3
  4   Name4
  5   Name5
  6   Name6
  7   Name7
  8   Name8
  9   Name9
 10  Name10
```

Data frames indexing

- Collection of variables → use $

```
students$ids
students[,1]
students[,"ids"]
```

Object - Lists

A collection of objects

It could consist of vectors, matrix, array, functions, and so on

```
parent_names <- c("Fred", "Mary")
number_of_children <- 2
child_ages <- c(4, 7, 9)
data.frame(parent_names, number_of_children, child_ages)
lst <- list(parent_names, number_of_children, child_ages)
lst[1]
lst[[1]]
class(lst[1])
class(lst[[1]])
lst[[1]][1]
lst[[1]][c(1,2)]
```

Data I/O

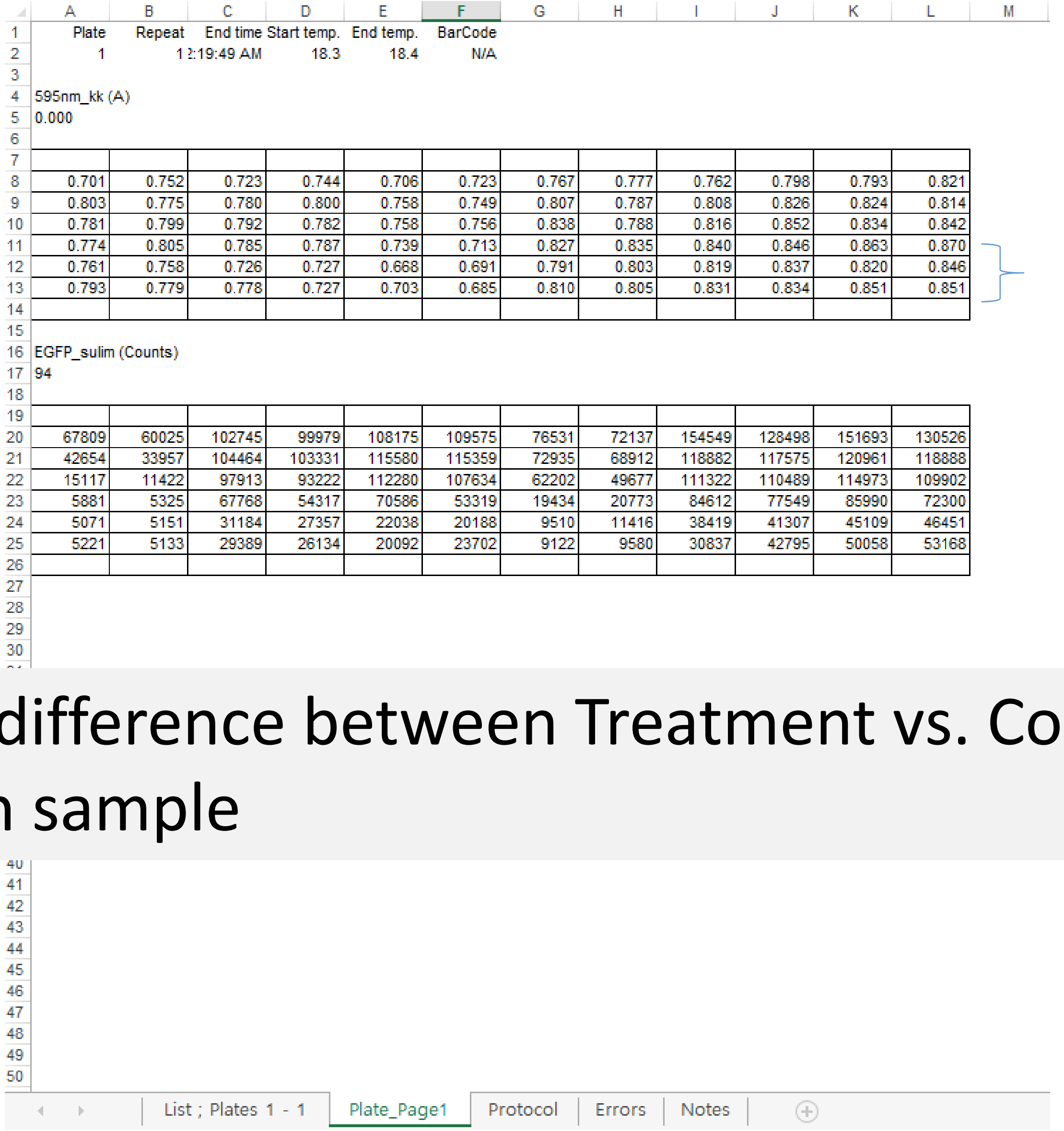
Text file write / read

```
x <- c(1,2,3,4)
y <- c(5,6,7,8)
xy<-data.frame(x=x, y=y)

write.table(xy, file="table_write.txt")
write.table(xy, file="table_write.txt", quote=F)
write.table(xy, file="table_write.txt", quote=F, row.names=F)
write.table(xy, file="table_write.txt", quote=F, row.names=F, sep=",")
write.table(xy, file="table_write.csv", quote=F, row.names=F, sep=",")
```

```
mydata<-read.table(file="table_write.csv")
mydata<-read.table(file="table_write.csv", sep=",")
mydata<-read.table(file="table_write.csv", sep=",", header=T)
plot(mydata$x, mydata$z)
```

Data read from excel files

Treatment

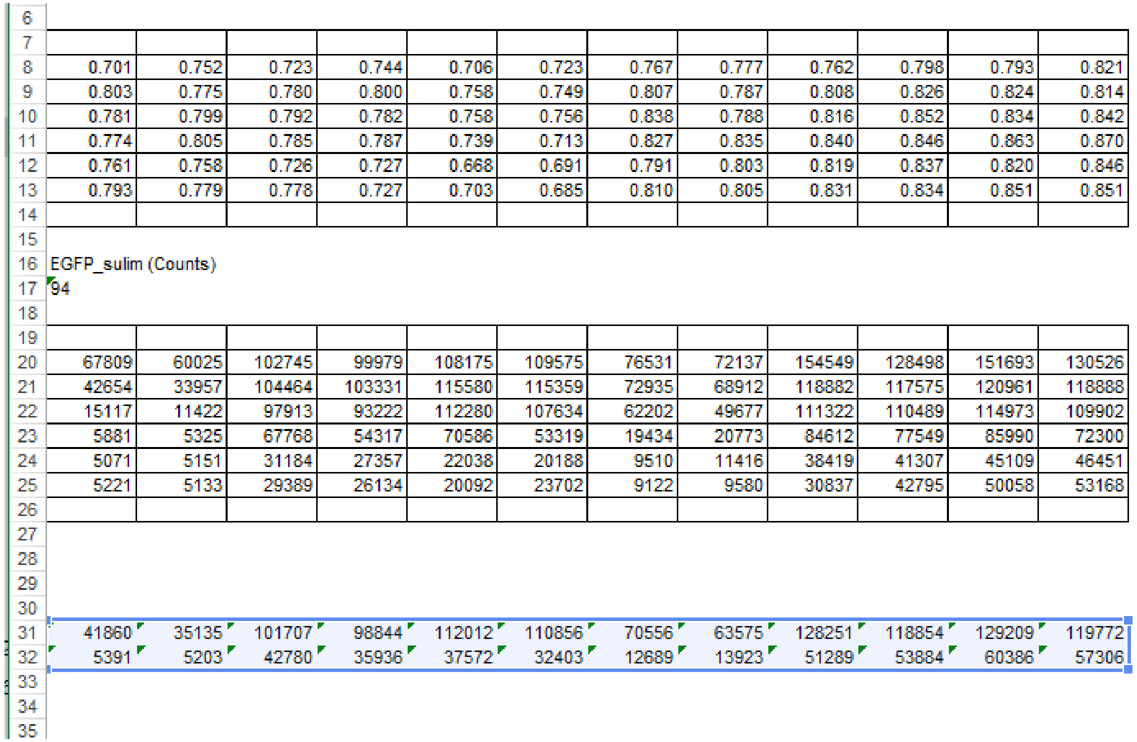


	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Plate	Repeat	End time	Start temp.	End temp.	BarCode							
2	1	1	2:19:49 AM	18.3	18.4	N/A							
3													
4	595nm_kk (A)												
5	0.000												
6													
7													
8	0.701	0.752	0.723	0.744	0.706	0.723	0.767	0.777	0.762	0.798	0.793	0.821	
9	0.803	0.775	0.780	0.800	0.758	0.749	0.807	0.787	0.808	0.826	0.824	0.814	
10	0.781	0.799	0.792	0.782	0.758	0.756	0.838	0.788	0.816	0.852	0.834	0.842	
11	0.774	0.805	0.785	0.787	0.739	0.713	0.827	0.835	0.840	0.846	0.863	0.870	
12	0.761	0.758	0.726	0.727	0.668	0.691	0.791	0.803	0.819	0.837	0.820	0.846	
13	0.793	0.779	0.778	0.727	0.703	0.685	0.810	0.805	0.831	0.834	0.851	0.851	
14													
15													
16	EGFP_sulim (Counts)												
17	94												
18													
19													
20	67809	60025	102745	99979	108175	109575	76531	72137	154549	128498	151693	130526	
21	42654	33957	104464	103331	115580	115359	72935	68912	118882	117575	120961	118888	
22	15117	11422	97913	93222	112280	107634	62202	49677	111322	110489	114973	109902	
23	5881	5325	67768	54317	70586	53319	19434	20773	84612	77549	85990	72300	
24	5071	5151	31184	27357	22038	20188	9510	11416	38419	41307	45109	46451	
25	5221	5133	29389	26134	20092	23702	9122	9580	30837	42795	50058	53168	
26													

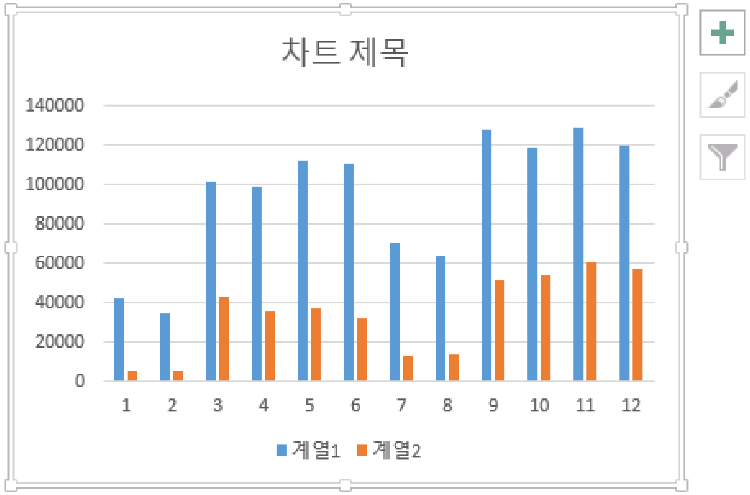
Control

Mean difference between Treatment vs. Control in each sample

Excel



6												
7												
8	0.701	0.752	0.723	0.744	0.706	0.723	0.767	0.777	0.762	0.798	0.793	0.821
9	0.803	0.775	0.780	0.800	0.758	0.749	0.807	0.787	0.808	0.826	0.824	0.814
10	0.781	0.799	0.792	0.782	0.758	0.756	0.838	0.788	0.816	0.852	0.834	0.842
11	0.774	0.805	0.785	0.787	0.739	0.713	0.827	0.835	0.840	0.846	0.863	0.870
12	0.761	0.758	0.726	0.727	0.668	0.691	0.791	0.803	0.819	0.837	0.820	0.846
13	0.793	0.779	0.778	0.727	0.703	0.685	0.810	0.805	0.831	0.834	0.851	0.851
14												
15												
16	EGFP_sulim (Counts)											
17	94											
18												
19												
20	67809	60025	102745	99979	108175	109575	76531	72137	154549	128498	151693	130526
21	42654	33957	104464	103331	115580	115359	72935	68912	118882	117575	120961	118888
22	15117	11422	97913	93222	112280	107634	62202	49677	111322	110489	114973	109902
23	5881	5325	67768	54317	70586	53319	19434	20773	84612	77549	85990	72300
24	5071	5151	31184	27357	22038	20188	9510	11416	38419	41307	45109	46451
25	5221	5133	29389	26134	20092	23702	9122	9580	30837	42795	50058	53168
26												
27												
28												
29												
30												
31	41860	35135	101707	98844	112012	110856	70556	63575	128251	118854	129209	119772
32	5391	5203	42780	35936	37572	32403	12689	13923	51289	53884	60386	57306
33												
34												
35												



readxl package

Download Rprog04-fl.xls from https://github.com/greendaygh/Rprog2019
Link: https://github.com/greendaygh/Rprog2019/raw/master/Rprog04-fl.xls
copy the file to working directory

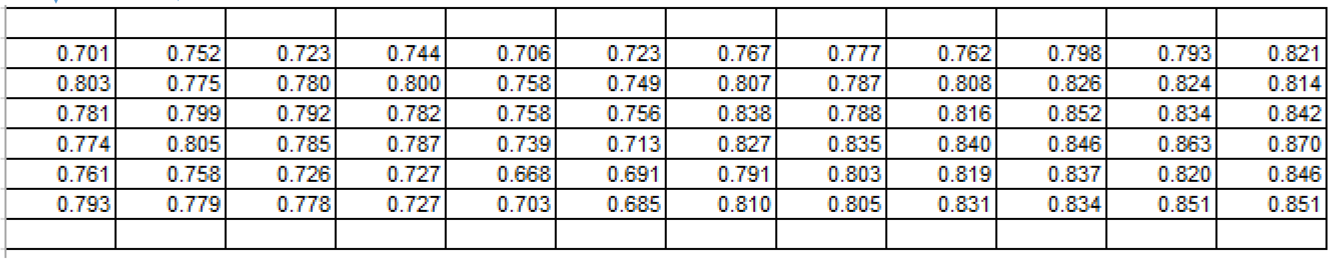
```
install.packages(readxl)
library(readxl)
dir()

mydata <- read_excel("Rprog04-fl.xls", sheet=2, skip = 6, col_names=F)

myod <- as.data.frame(mydata[1:9, ])
mygfp <- as.data.frame(mydata[12:21, ])

myod[,1] <- as.numeric(myod[,1])
mygfp[,1] <- as.numeric(mygfp[,1])
```

모든 경우 그렇지는 않으며
해당 excel file 특이적으로 첫번째
컬럼이 character임

** tibble vs. data.frame
- never changes type of the inputs
- never changes name of variables
- never create rownames

Data manipulation

Sample1, Sample2, ...



Treat

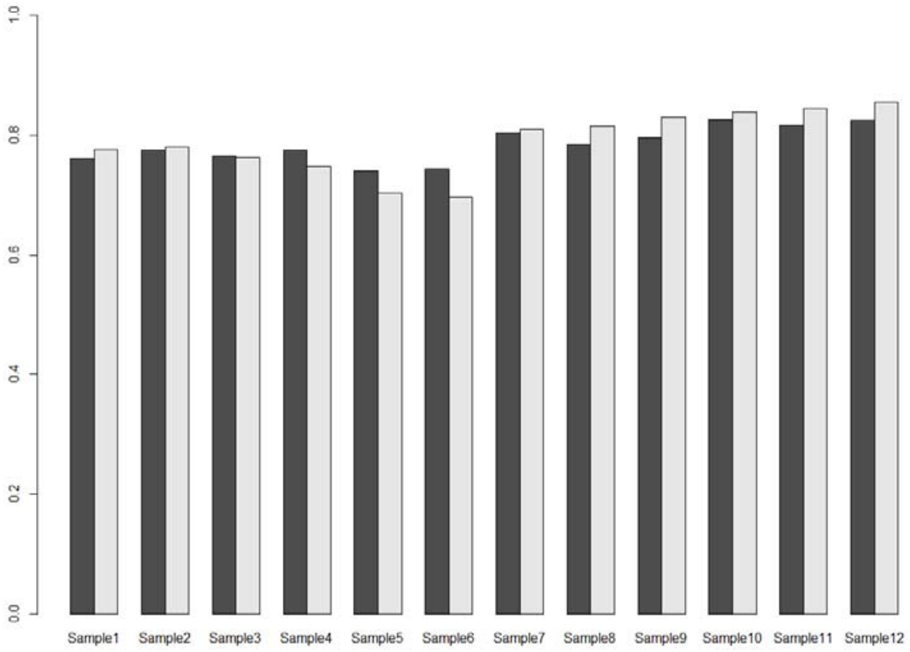
Repeat1	0.701	0.752	0.723	0.744	0.706	0.723	0.767	0.777	0.762	0.798	0.793	0.821
Repeat2	0.803	0.775	0.780	0.800	0.758	0.749	0.807	0.787	0.808	0.826	0.824	0.814
Repeat3	0.781	0.799	0.792	0.782	0.758	0.756	0.838	0.788	0.816	0.852	0.834	0.842
	0.774	0.805	0.785	0.787	0.739	0.713	0.827	0.835	0.840	0.846	0.863	0.870
	0.761	0.758	0.726	0.727	0.668	0.691	0.791	0.803	0.819	0.837	0.820	0.846
	0.793	0.779	0.778	0.727	0.703	0.685	0.810	0.805	0.831	0.834	0.851	0.851

Control

```
# OD
myod_treat <- myod[2:4,]
myod_control <- myod[5:7,]

sample_names <- paste("Sample", c(1:12), sep="")
replicate_labels <- paste("Rep", c(1:3), sep="")

rownames(myod_treat) <- replicate_labels
colnames(myod_treat) <- sample_names
rownames(myod_control) <- replicate_labels
colnames(myod_control) <- sample_names
```

Mean comparison

```
mean_treat <- colMeans(myod_treat)
mean_control <- colMeans(myod_control)

plot(mean_treat, type="h")
barplot(mean_treat, ylim=c(0,1))

mean_test <- data.frame(mean_treat, mean_control)
barplot(t(mean_test), ylim=c(0,1), beside=T)
```



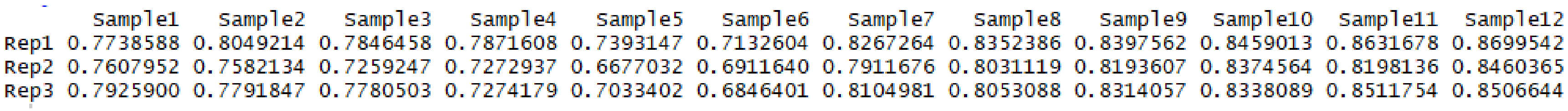
Exercise 3-2) standard deviation

- Make a function for computing standard deviations
  - Function name: mysd
  - Input parameter: inx (numeric vector, length(inx)>1)
  - Return: standard deviation of inx
- Generate a variable x with 10 random numbers using 'sample' from 1 to 100
- Use mysd to compute the standard deviation of x

$$sample\ mean = \ \bar{x} = \frac{1}{n}(x_1 + x_2 + \ \ldots + x_n) = \frac{1}{n}\sum_i x_i$$

$$sample\ standard\ deviation = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_i (x_i - \bar{x}\ )^2}$$

Apply



```
       Sample1   Sample2   Sample3   Sample4   Sample5   Sample6   Sample7   Sample8   Sample9  Sample10  Sample11  Sample12
Rep1 0.7738588 0.8049214 0.7846458 0.7871608 0.7393147 0.7132604 0.8267264 0.8352386 0.8397562 0.8459013 0.8631678 0.8699542
Rep2 0.7607952 0.7582134 0.7259247 0.7272937 0.6677032 0.6911640 0.7911676 0.8031119 0.8193607 0.8374564 0.8198136 0.8460365
Rep3 0.7925900 0.7791847 0.7780503 0.7274179 0.7033402 0.6846401 0.8104981 0.8053088 0.8314057 0.8338089 0.8511754 0.8506644
```

```
apply(myod_control, 1, mean)
apply(myod_control, 2, mean)
```

1: row (가로)
2: column (세로)

function

```
apply(myod_control, 2, function(x){
  xmean <- mean(x)
  return(xmean)
})
```

```
apply(myod_control, 2, sd)
```

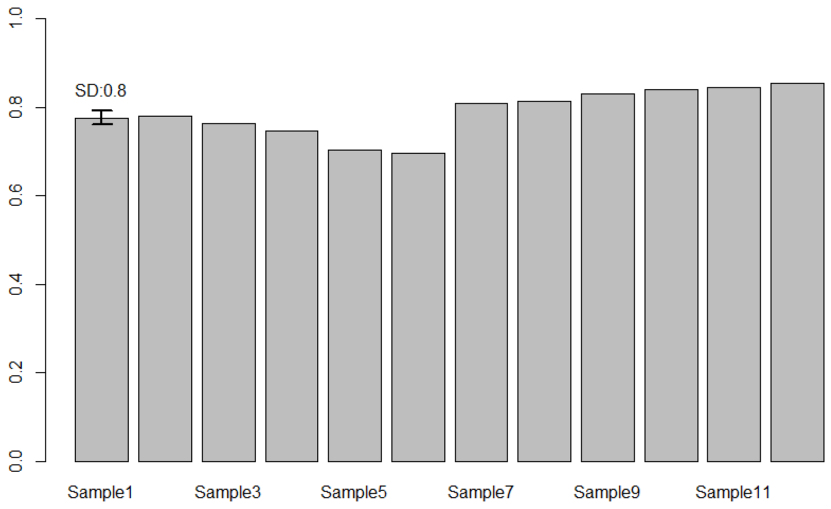
The apply function

- For matrices, vectorized functions are applied to each element

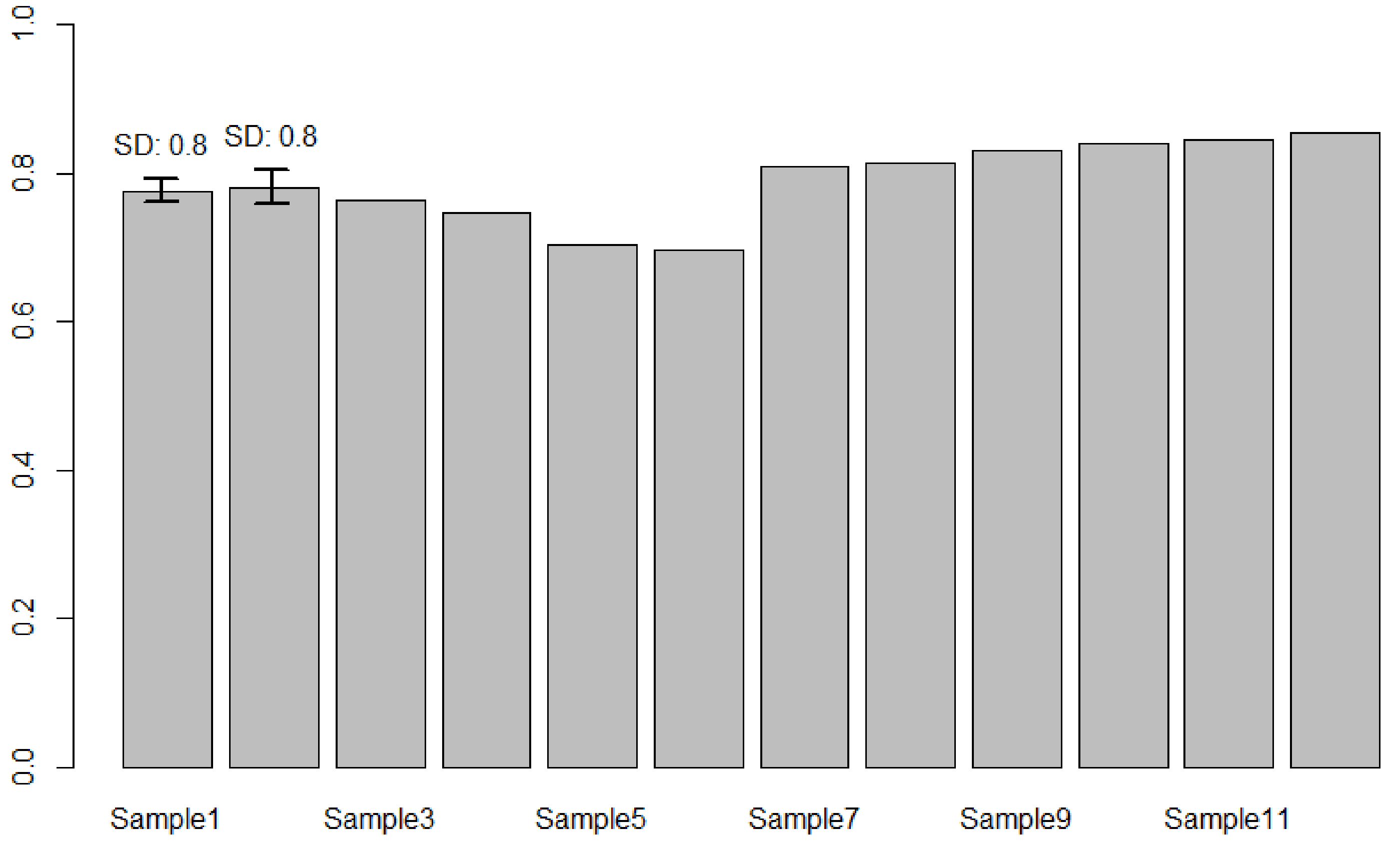
```
# a common pattern to generate matrix
rowSums(m)
colSums(m)
apply(m, 1, mean)
apply(m, 2, mean)
sapply(m, sum)

?sweep
sweep(m, 1, 10)
sweep(m, 1, 10, "+")
sweep(m, 1, 10, "/")
```

barplot with sd

```
control_mean <- apply(myod_control, 2, function(x){mean(x)})
control_sd <- apply(myod_control, 2, mysd)

barplot(control_mean, width=0.8, space=0.2, col="gray", ylim=c(0,1))
arrows(0.5, control_mean[1], 0.5, control_mean[1]+control_sd[1], length=0.1, angle=90)
arrows(0.5, control_mean[1], 0.5, control_mean[1]-control_sd[1], length=0.1, angle=90)
lab <- paste("SD:", round(control_mean[1]+control_sd[1],1))
text(0.5, control_mean[1]+control_sd[1]+0.05, labels = lab)
```



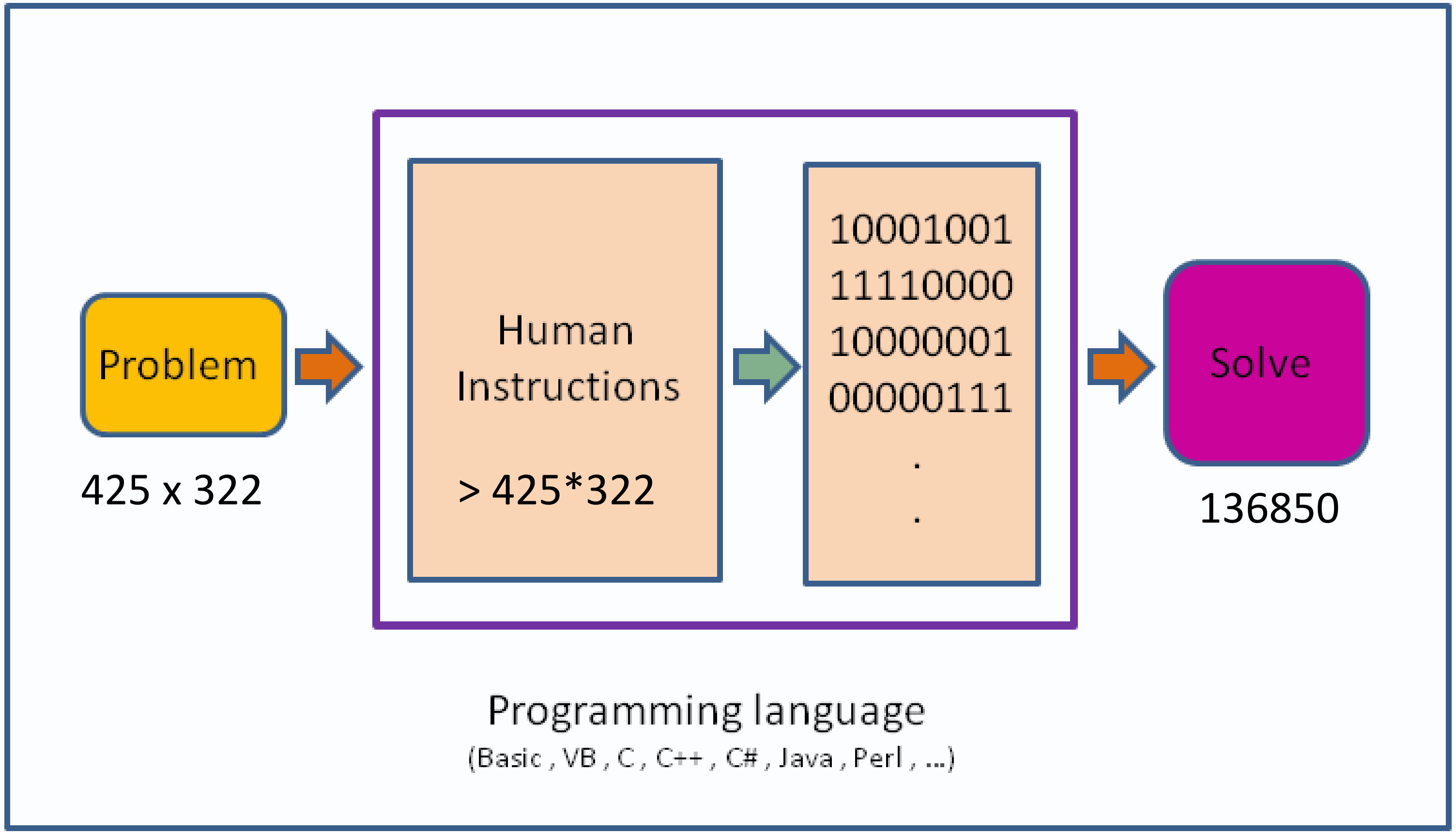
Exercise 3-3) barplot with sd



Programming

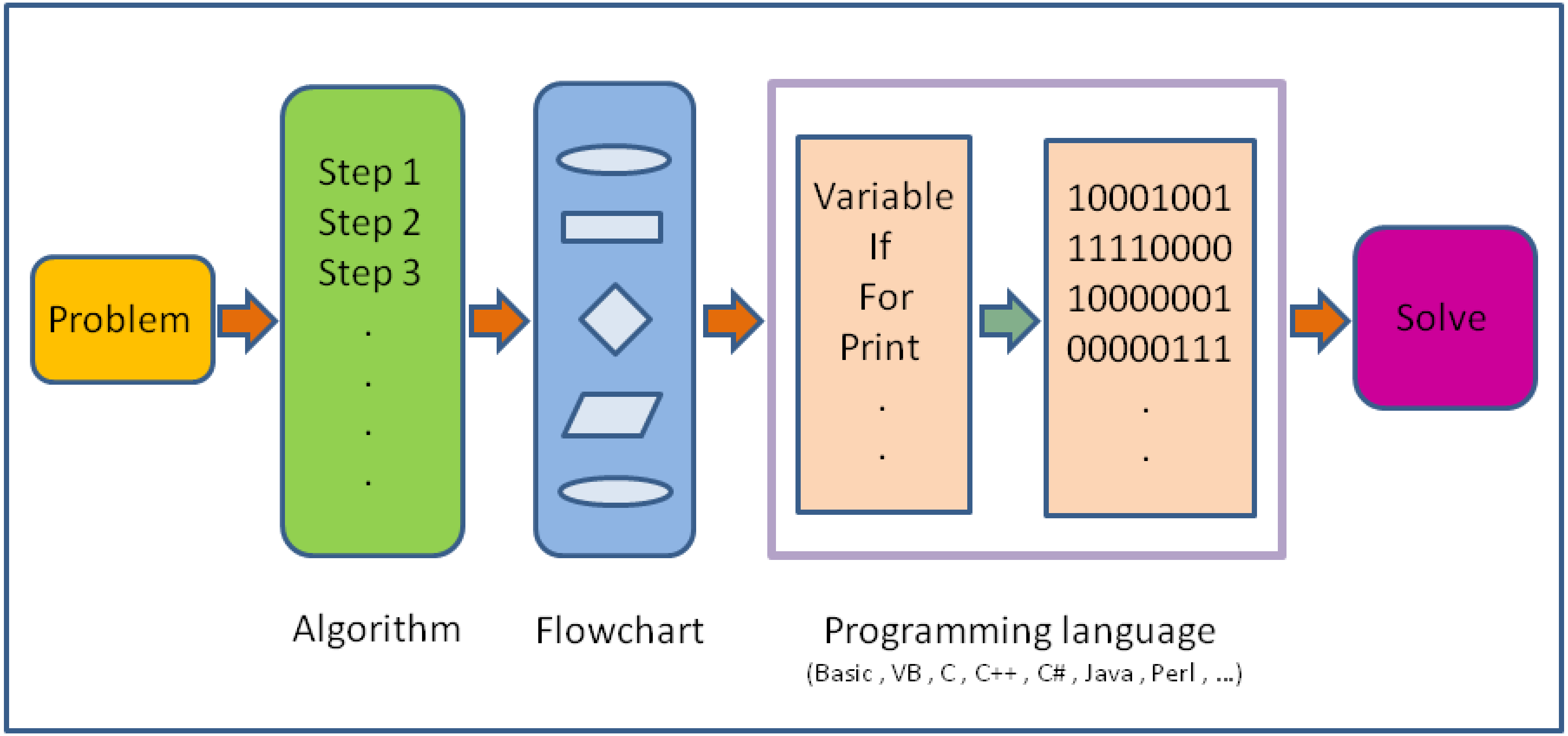
if, if else, for, while, break

What is a programming language?



https://en.wikiversity.org/wiki/Introduction_to_Programming/About_Programming

What is a programming language?



Problem

Step 1
Step 2
Step 3
.
.
.
.

Variable
If
For
Print
.
.

10001001
11110000
10000001
00000111
.
.

Solve

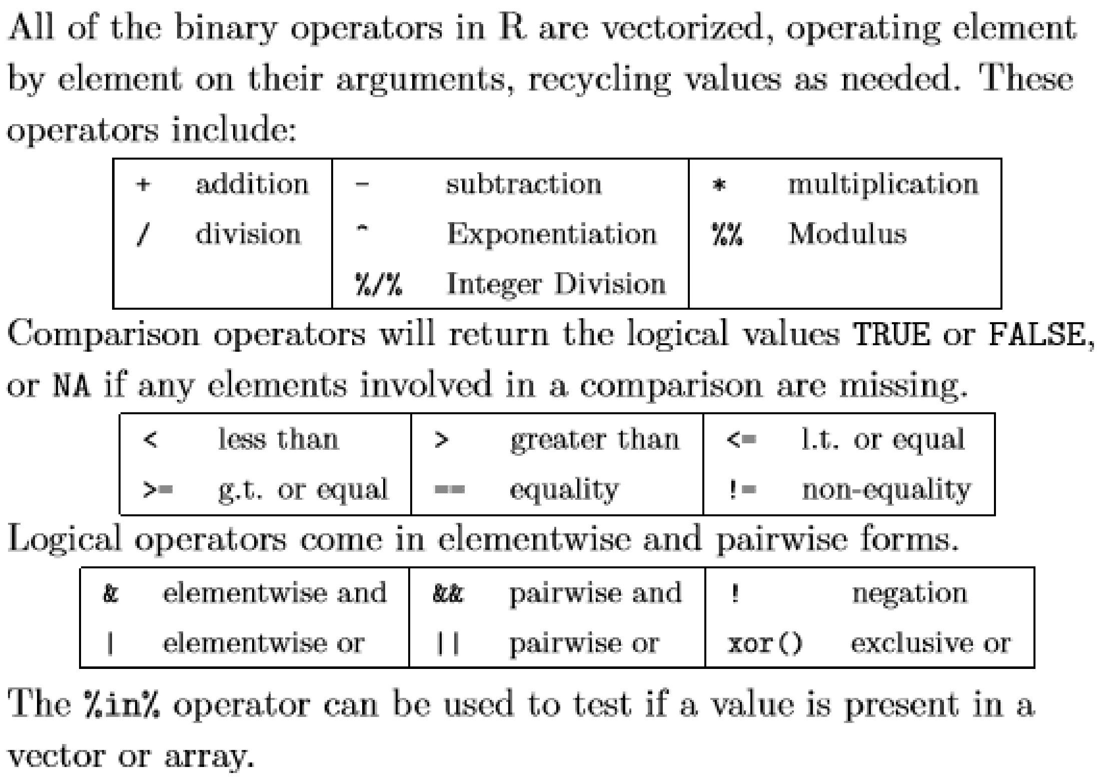
Algorithm

Flowchart

Programming language
(Basic , VB , C , C++ , C# , Java , Perl , ...)

if / if else

```
if(condition) {
       expression
}


if(condition) {
       expression1
} else {
       expression2
}


if(condition) {
       expression1
} else if (condition2) {
       expression2
} else {
       expression3
}
```



All of the binary operators in R are vectorized, operating element by element on their arguments, recycling values as needed. These operators include:

+	addition	-	subtraction	*	multiplication
/	division	^	Exponentiation	%%	Modulus
		%/%	Integer Division		

Comparison operators will return the logical values TRUE or FALSE, or NA if any elements involved in a comparison are missing.

<	less than	>	greater than	<=	l.t. or equal
>=	g.t. or equal	==	equality	!=	non-equality

Logical operators come in elementwise and pairwise forms.

&	elementwise and	&&	pairwise and	!	negation
|	elementwise or	||	pairwise or	xor()	exclusive or

The %in% operator can be used to test if a value is present in a vector or array.

if / if else

```
x <- 5
if(x > 0){
   print("Positive number")
}

if(x > 0) print("Positive number")



x <- -5
if(x > 0){
       print("Non-negative number")
} else {
       print("Negative number")
}

if(x > 0)
       print("Non-negative number")
else
       print("Negative number")
```

for loop

```
for(val in sequence){
    statement
}
```

i ← value

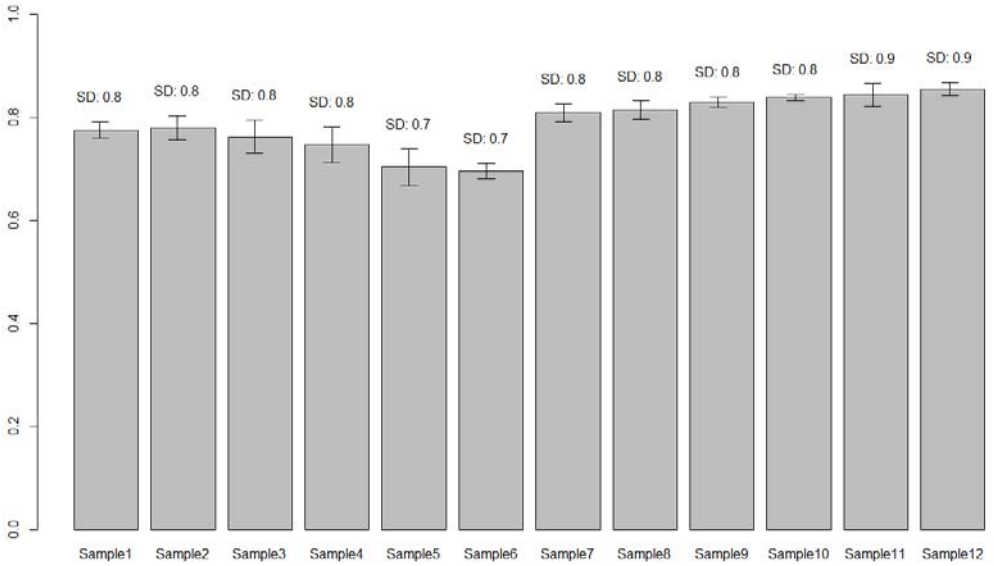
```
x <- c(2,5,3,9,8,11,6)
count <- 0
for(i in x) {
    if(i %% 2 == 0)  count <- count+1
}
print(count)
```

i ← index

```
x <- c(2,5,3,9,8,11,6)
count <- 0
for(i in 1:length(x)) {
       val <- x[i]
    if(val > mean(x))  count <- count+1
}
print(count)
cat("count:", count, "\n")
```

Exercise 3-4) for

```
barplot(control_mean, width=0.83, space=0.2, col="gray", ylim=c(0,1))

arrows(0.5, control_mean[1], 0.5, control_mean[1]+control_sd[1], length=0.1, angle=90)
arrows(0.5, control_mean[1], 0.5, control_mean[1]-control_sd[1], length=0.1, angle=90)
lab <- paste("SD:", round(control_mean[1]+control_sd[1],1))
text(0.5, control_mean[1]+control_sd[1]+0.05, labels = lab)

arrows(1.5, control_mean[2], 1.5, control_mean[2]+control_sd[2], length=0.1, angle=90)
arrows(1.5, control_mean[2], 1.5, control_mean[2]-control_sd[2], length=0.1, angle=90)
lab <- paste("SD:", round(control_mean[2]+control_sd[2],1))
text(1.5, control_mean[2]+control_sd[2]+0.05, labels = lab)
```

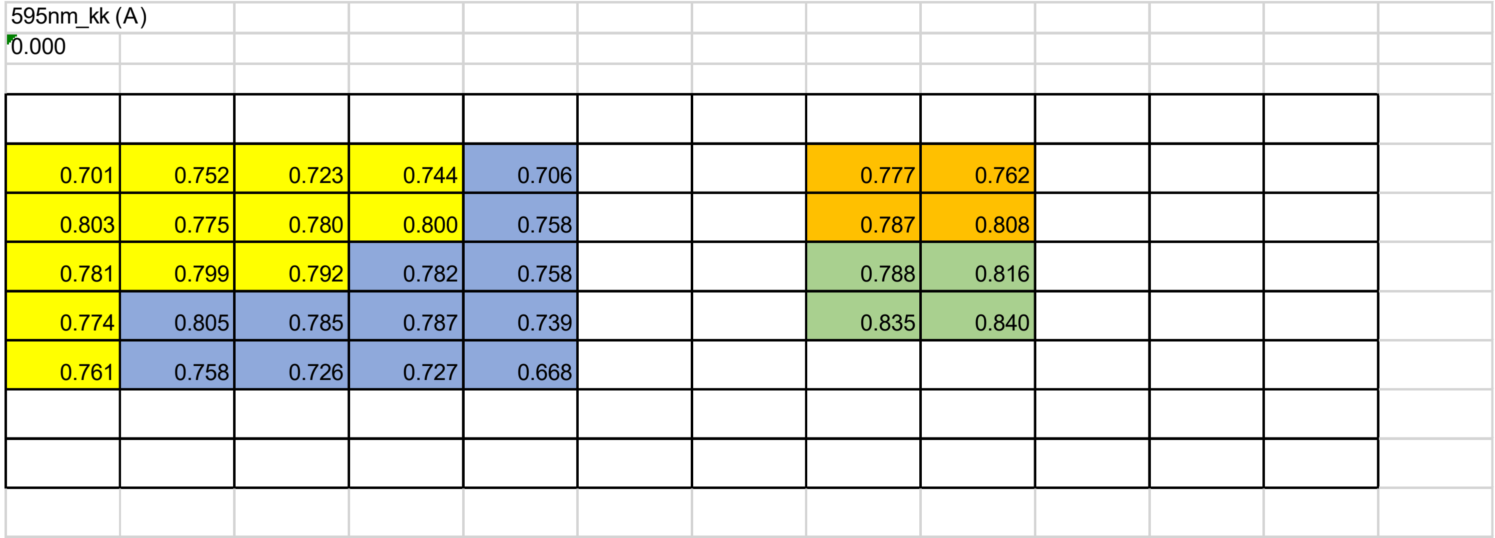


Exercise 3-5) Writing a function for..

- read excel file (case, control)
- compare means
  - compute difference in each sample
  - plot bar graph
- return mean differences as a numeric vector

```
mean_comparison <- function(filename, plot_flag){
  # read excel file
  # get case control matrix (OD values only)
  # compute mean, sd
  # barplot
  # return mean differences
}
```

What if..



595nm_kk (A)

0.000

0.701	0.752	0.723	0.744	0.706			0.777	0.762			
0.803	0.775	0.780	0.800	0.758			0.787	0.808			
0.781	0.799	0.792	0.782	0.758			0.788	0.816			
0.774	0.805	0.785	0.787	0.739			0.835	0.840			
0.761	0.758	0.726	0.727	0.668							

Next

- R programming
  - maze
- Data manipulation
  - dplyr