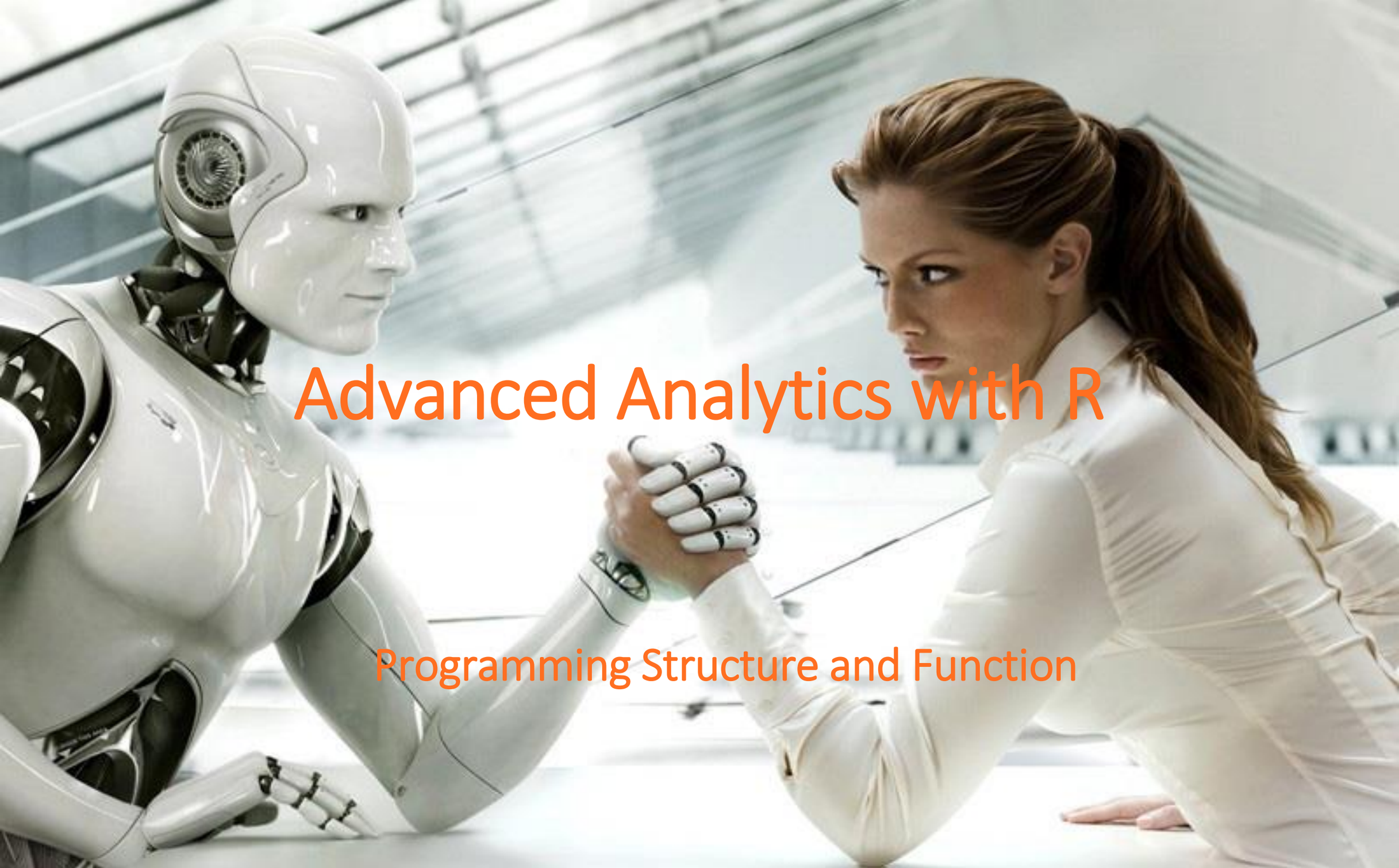


Advanced Analytics with R

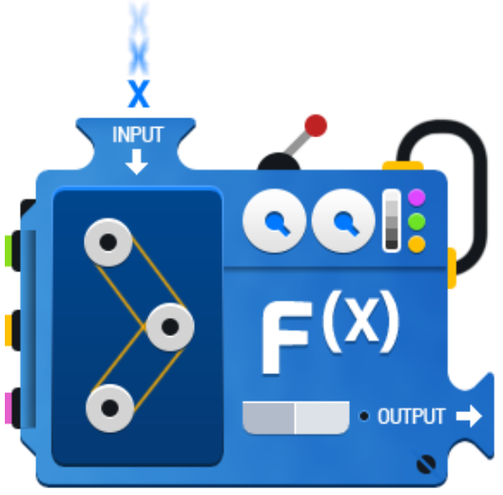
Programming Structure and Function

What is programming?

What do the following functions have in common?

- paste()
- sum()
- dir()
- dim()
- getwd()

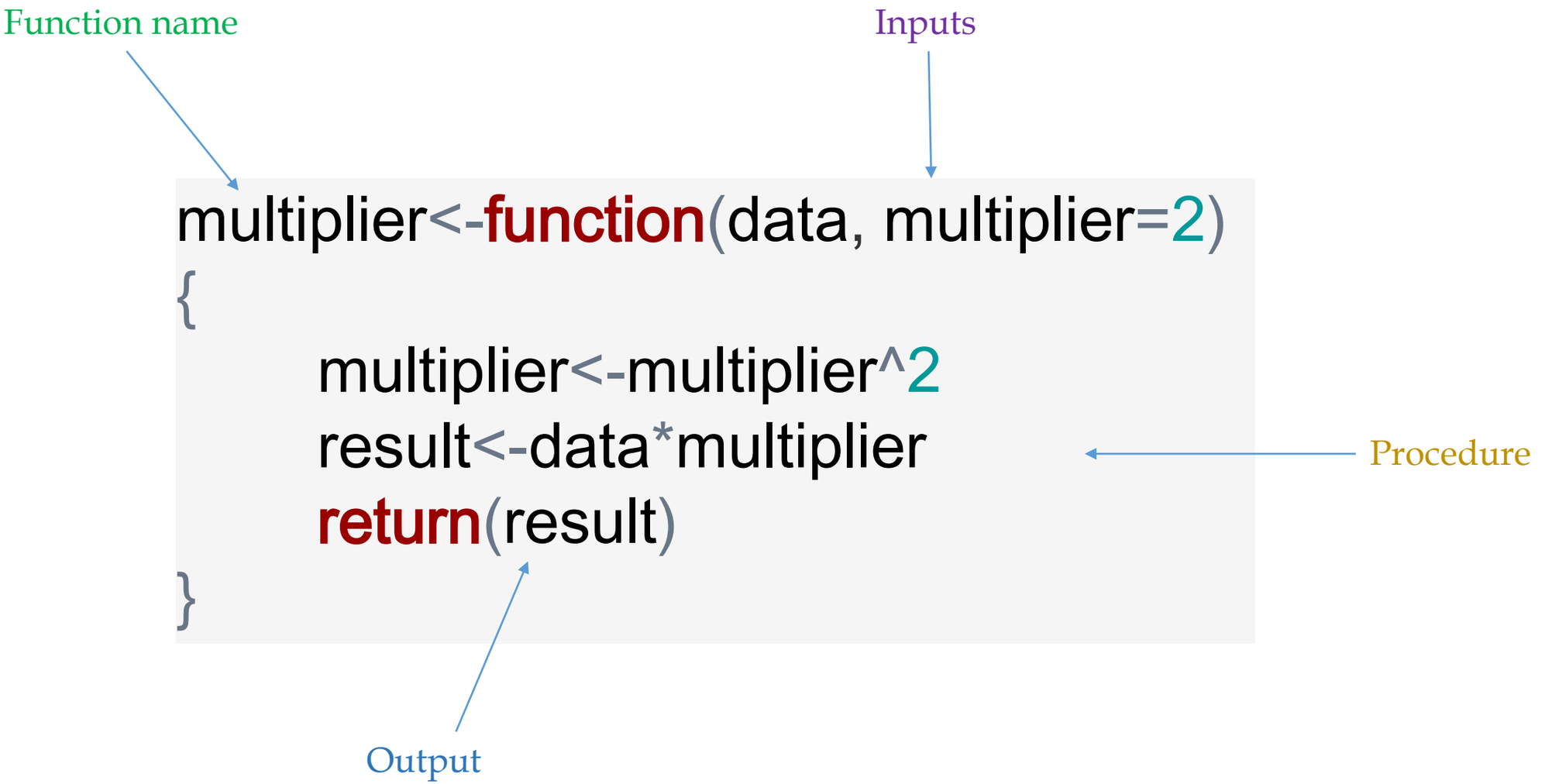
What is function?



Designed to perform a specific task

Can be used repeatedly

Very often, a function will take in some inputs and generate some outputs

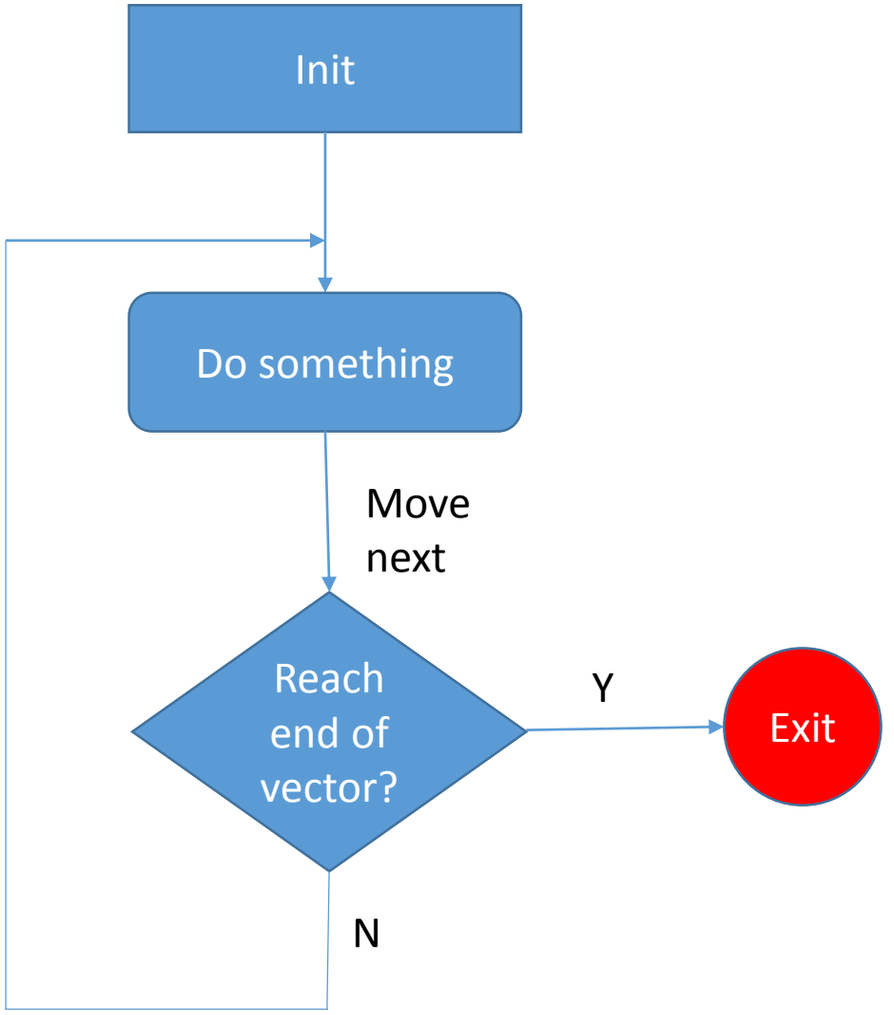

Function name
Inputs
multiplier<-function(data, multiplier=2)
{
multiplier<-multiplier^2
result<-data*multiplier
return(result)
}
Procedure
Output

Reuse a function

- Write function in a R script.

- Load the R script when needed.

Programming Structure – *for* loop



Programming Structure – *for* loop

```
#create a vector of 20 random numbers
a <- rnorm(20)
a

#square the first 10 numbers
for(i in 1:10)
{
   a[i] <- a[i] * a[i]
}
a
```

Exercise

- What if I want to square the numbers with odd indices?

Exercise

- Produce the first 100 numbers in the famous Fibonacci sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Nested *for* loops

```
# Create a 5 x 5 matrix
m <- matrix(nrow=5, ncol=5)

# For each row and for each column, assign values based on position: difference of the two
indices
for(i in 1:dim(m)[1]) {
   for(j in 1:dim(m)[2]) {
      m[i,j] = i - j
   }
}

# show m
m
```

Notes

- *for* loop is not very efficient. When possible, try to use *apply* functions.

```
v<-rnorm(1000000)

system.time(for(i in 1:length(v))  v[i] <- v[i]*v[i])

system.time(v <- sapply(v, function(x) x^2))
```

Convert nested for loops using *sapply*

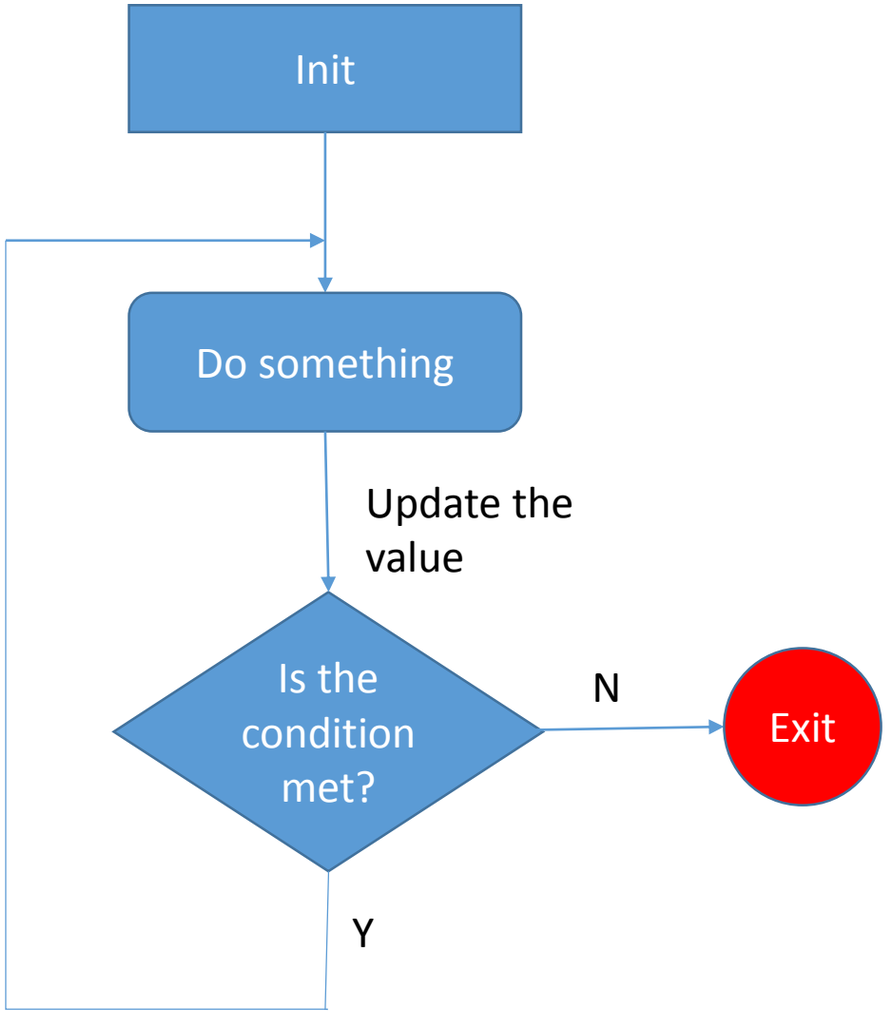
```
m <- matrix(nrow=5, ncol=5)

f <- function(data=m, i, j) m[i,j] <- i-j

sapply(1:5, function(i) sapply(1:5, function(j) f(m, i, j)))
```

for and *get*

R does not directly support iteration over nonvector sets, but there are indirect yet easy ways to accomplish it. One way would be to use **lapply**(), as shown in Section 5.8. Another would be to use **get**(), as in the following example. Here we have two matrices, **u** and **v**, containing statistical data, and we wish to apply R linear regression function **lm**() to each of them:

```
u <- matrix (1:10, nrow = 2, ncol = 5)
v <- matrix (10:23, nrow = 7, ncol=2)
for(m in c("u","v")) {
   z <- get(m)
   print(z[1,])
}
```

Programming Structure – ***while*** loop



Programming Structure – *while* loop

```
#create a vector of 20 random numbers
a <- rnorm(20)
a

i<-1
#square the first 10 numbers
while(i < 11)
{
   a[i] <- a[i] * a[i]

   i<-i+1
}
a
```

Exercise

- Convert your previous codes from for loop to while loop.

while is more flexible than *for*

```
> i <- 1
> while(1) {
+     i <- i+4
+     if (i > 10) break
+ }
> i
[1] 13
```

Exercise

- Write a function with *n* as the parameter that it will print Fibonacci sequence until the value is more than *n*.

if-else

```
x<-c(3,10,15,2)
f<-function(x)
{
   for(i in 1:length(x))
   {
      #if the elment is divisible by 5, then divide it by 5. otherwise multiply it by 5
      if(x[i] %% 5==0)
      {
         x[i]<-x[i]/5
      }
      else
      {
         x[i]<-x[i]*5
      }
   }
   return(x)
}
f(x)
```

Case Study – CEO Compensation