Create your own functions - much simpler!

A function is defined as follows.

```
> quadx <- function(x) {          # function definition, input variables
    x^2 + x + 1                 # R statements, what the function does
  }
```

The return value is the last value computed -- but you can also use the "return" function.

```
> quadx <- function(x) {
  return( x^2 + x + 1 )
}
> fxy <- function(x, y=3) { x+y^2 }
```

Arguments can have default values.

When you call a function and list the argument **names**, you don't have to worry about the order you list them in (this is very useful for functions that expect many arguments -- in particular arguments with default values).

```
> f(y=4, x=3.14)
> f(4, 3.14)
```

without argument names, it will assign x=4, y=3.14

Create your own functions - much simpler!

After the arguments, in the definition of a function, you can put three dots represented the arguments that have not been specified and that can passed through another function (very often, the "plot" function).

```
f <- function(x, ...) {          # ... you could put here col="red", pch=19, or any
  plot(x, ...)                   # other valid argument
}
```

But you can also use this to write functions that take an arbitrary number of arguments:

```
f <- function (...) {
  query <- paste(...)  # Concatenate all the arguments to form a string
  con <- dbConnect(dDriver("SQLite"))
  dbGetQuery(con, query)
  dbDisconnect(con)
}

f <- function (...) {
  l <- list(...)       # Put the arguments in a (named) list
  for (i in seq(along=l)) {
    cat("Argument name:", names(l)[i], "Value:", l[[i]], "\n")
  }
}
```

Functions have NO SIDE EFFECTS: all the modifications are local. In particular, you cannot write a function that modifies a global variable.

Creating Matrices

dim function sets the dimensions of an object, can be used to convert vector to matrix

```
> value <- rnorm(6)
> dim(value) <- c(2,3)
> value
[,1] [,2] [,3]
[1,] 0.7093460 -0.8643547 -0.1093764
[2,] -0.3461981 -1.7348805 1.8176161
```

To convert back to a vector, set dim to NULL

```
> dim(value) <- NULL
```

matrix function also converts a vector to a matrix

```
> matrix(value,2,3)

[,1] [,2] [,3]
[1,] 0.7093460 -0.8643547 -0.1093764
[2,] -0.3461981 -1.7348805 1.8176161
```

If we want to fill by rows instead:

```
> matrix(value,2,3,byrow=T)
[,1] [,2] [,3]
[1,] 0.709346 -0.3461981 -0.8643547
[2,] -1.734881 -0.1093764 1.8176161
```

rbind/cbind: "binds" or pastes rows/columns together to form a matrix (must be same length). Can also add rows or columns to existing matrix.

```
> value <- matrix(rnorm(6),2,3,byrow=T)
> value2 <- rbind(value,c(1,1,2))
> value2
[,1] [,2] [,3]
[1,] 0.5037181 0.2142138 0.3245778
[2,] -0.3206511 -0.4632307 0.2654400
[3,] 1.0000000 1.0000000 2.0000000
> value3 <- cbind(value2,c(1,1,2))
```

Creating Data Frames

data.frame converts a matrix or collection of vectors into a data frame

```
> value3 <- data.frame(value3)
> value3
   X1 X2 X3 X4
1 0.5037181 0.2142138 0.3245778 1
2 -0.3206511 -0.4632307 0.2654400 1
3 1.0000000 1.0000000 2.0000000 2
```

Join two columns of data together:

```
> value4 <- data.frame(rnorm(3),runif(3))
> value4
    rnorm.3. runif.3.
1 -0.6786953 0.8105632
2 -1.4916136 0.6675202
3  0.4686428 0.6593426
```

Row and column names are already assigned to a data frame but they may be changed using the names and row.names functions.

```
> names(value3)    # show column names
[1] "X1" "X2" "X3" "X4"
> row.names(value3)   # show row names
[1] "1" "2" "3"
```

To change names:

```
> names(value3) <- c("C1","C2","C3","C4")
> row.names(value3) <- c("R1","R2","R3")
```

Names can also be specified on creation:

```
>data.frame(C1=rnorm(3),C2=runif(3),row.names=c("R1","R2","R3")
C1 C2
R1 -0.2177390 0.8652764
R2 0.4142899 0.2224165
R3 1.8229383 0.5382999
```

Accessing Elements of Vector or Matrix

Accessing elements is achieved through *indexing*.

by:

•a vectorof positive integers: inclusion

• a vector of negative integers: exclusion

• a vector of logical values: T= in F = out

• a vector of names:

For the latter, if a zero index occurs on the right, no element is selected. If a zero index

occurs on the left, no assignment is made. An empty index position stands for the lot!

Indexing Vectors

Produce random sample of values between one and five, twenty times:

```
> x <- sample(1:5, 20, rep=T)
> x
[1] 3 4 1 1 2 1 4 2 1 1 5 3 1 1 1 2 4 5 5 3
> x == 1   # which equal 1?
[1] FALSE FALSE TRUE TRUE FALSE TRUE
    FALSE FALSE TRUE
[10] TRUE FALSE FALSE TRUE TRUE TRUE
    FALSE FALSE FALSE
[19] FALSE FALSE
> ones <- (x == 1) # parentheses unnecessary
> # "ones" is an index vector, which =1?

> x[ones] <- 0   # set x=1 to x=0
> x
[1] 3 4 0 0 2 0 4 2 0 0 5 3 0 0 0 2 4 5 5 3
```

Accessing Elements of Vector or Matrix

```
> others <- (x > 1) # index vector
>     # indicating which x's are > 1
> y <- x[others]   # take x>1 and save as
>                  # new vector y
> y      # y is shorter because <1 are dropped
[1] 3 4 2 4 2 5 3 2 4 5 5 3
```

Which are greater than 1?

```
> which(x > 1)
```

```
[1] 1 2 5 7 8 11 12 16 17 18 19 20
```

These are the index values (the number that indicates the position in the vector)

Indexing Data Frames

by either row or column using a specific name (row or column name) or a number.

```
> value3
C1 C2 C3 C4
R1 0.5037181 0.2142138 0.3245778 1
R2 -0.3206511 -0.4632307 0.2654400 1
R3 1.0000000 1.0000000 2.0000000 2
> value3[, "C1"] <- 0   # by col name
> value3
C1 C2 C3 C4
R1 0 0.2142138 0.3245778 1
R2 0 -0.4632307 0.2654400 1
R3 0 1.0000000 2.0000000 2
```

Indexing by row (name):

```
> value3["R1", ] <- 0
> value3
C1 C2 C3 C4
R1 0 0.0000000 0.0000000 0
R2 0 -0.4632307 0.2654400 1
R3 0 1.0000000 2.0000000 2
```

```
> value3[ ] <- 1:12
> value3
C1 C2 C3 C4
R1 1 4 7 10
R2 2 5 8 11
R3 3 6 9 12
```

To access the first two rows (by row #) :

```
> value3[1:2,]
C1 C2 C3 C4
R1 1 4 7 10
R2 2 5 8 11
```

```
> value3[,1:2]  # first two columns
C1 C2
R1 1 4
R2 2 5
R3 3 6
```

To access all matrix elements > 5, use subsetting and logical operators

```
> as.vector(value3[value3>5])
[1] 6 7 8 9 10 11 12
```

Indexing Data Frames

Also, the subset() function. This is often preferred for unambiguousness.

Lists

Like a data frame, but more irregular. Lists can be composed of vectors, each of different type and length.

Often used for returning output.

Created using **list()**

```
> L1 <- list(x = sample(1:5, 20, rep=T),  y = rep(letters[1:5], 4), z = rpois(20, 1))
> L1
$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
$y
[1] "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a" "b"
[13] "c" "d" "e" "a" "b" "c" "d" "e"
$z
[1] 1 3 0 0 3 1 3 1 0 1 2 2 0 3 1 1 0 1 2 0
```

Accessing List Elements

By **name** of that component (if names are assigned) or by **number**

- using subsetting ([[ ]]) NOTE: Double brackets
- the extraction operator ($).

```
> L1[["x"]]
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
> L1$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
> L1[[1]]   # extracts the first list element (here, x)
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
```

To extract a sublist (i.e., part of a list as opposed to a vector which you can think of as a single component of a list), we use single brackets. The following example extracts the first component only.

```
> L1[1]
$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
```

Working with Lists

Length:

```
> length(L1)  #  is number of components

[1] 3
```

Names of lists can also be altered in a similar way to that for data frames.

```
> names(L1) <- c("Item1","Item2","Item3")
```

Above assigns names to elements of L1.

Indexing: also similar to data frames:

```
> L1$Item1[L1$Item1>2]  # L1$item1's > 2

[1] 4 3 4 5 3 3 3 5 3 3 5
```

Joining two lists: c() or append():

```
> L2 <- list(x=c(1,5,6,7),
+  y=c("apple","orange","melon","grapes"))
> c(L1,L2)

$Item1
[1] 2 4 3 4 1 5 3 1 1 2 3 3 5 2 1 3 2 3 5 1

$Item2
[1] "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a" "b"
[13]"c" "d" "e" "a" "b" "c" "d" "e"

$Item3
[1] 0 0 2 1 1 0 2 0 0 1 1 1 0 0 1 1 1 3 0 2

$x
[1] 1 5 6 7

$y
[1] "apple" "orange" "melon" "grapes"
```

Working with Lists

Append Function:

```
> append(L1,L2,after=2)

$Item1
[1] 2 4 3 4 1 5 3 1 1 2 3 3 5 2 1 3 2 3 5 1

$Item2
[1] "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a"
[12]"b" "c" "d" "e" "a" "b" "c" "d" "e"

$x
[1] 1 5 6 7

$y
[1] "apple" "orange" "melon" "grapes"

$Item3
[1] 0 0 2 1 1 0 2 0 0 1 1 1 0 0 1 1 1 3 0 2
```

Adding elements to a list can be achieved by

- adding a new component name:

```
> L1$Item4 <-
    c("apple","orange","melon","grapes")

# alternative way

> L1[["Item4"]] <-
    c("apple","orange","melon","grapes")
```

- adding a new component element, whose index is greater than the length of the list

```
L1[[4]] <-
    c("apple","orange","melon","grapes")

> names(L1)[4] <- c("Item4")
```