Indexing is the key to working with and manipulating R data. There are three ways to index data in R:

- position
- logical vector
- name

Run the following to see an example of each type of indexing.

```r
> x <- c(chol = 234, sbp = 148, dbp = 78, age = 54)
> x
> x[1] # by position
> x[x>150]# by logical
> x["chol"] # by name
```

You can use indexing to replace or change a data entry.

```r
> x[1] <- 250 #by position
> x[x<100] <- NA # by logical
> x["sbp"] <- 150 # by name
> x
```

Let’s look at the three approaches to indexing in a bit more detail.

1 indexing vectors

1.1 by position

```r
> x<-1:40
> x[11] #only the 11th element
> x[-11] #exclude the 11th element
```
1.2 by logical

R uses the following symbols to establish logical relationships between variables.

== IS equivalent to
! is NOT
& AND
| OR (if either or both comparison elements are TRUE)
xor EITHER (element-wise exclusive or operator, if either, but not both, comparison elements TRUE)
&& || special operators, control flow in "if" functions, only the first element of logical is used.

In addition, the `which()` function returns an integer vector of positions from a Boolean operation, for example

```r
> age <- c(8, NA, 7, 4)
> which(age<5 | age>=8)
```

Here, the positions 1 and 4 in the vector "age" meet the Boolean definition.

To use a logical expression to index R data:

1. create a logical vector
2. use the logical vector to index data

Let's take a look at an example. First create three vectors of related data.

```r
> names<-c("dopey", "grumpy", "doc", "happy", "bashful", + "sneezy", "sleepy")
> ages<-c(142, 240, 232, 333, 132, 134, 127)
> sex<-c("m", "m", "f", "f", "f", "m", "m")
```

Now, do some indexing.

```r
> young <- ages < 150 #create logical vector
> names[young] #index name vector using logical vector
> names[!young] # old dwarves
> male<- sex == "m" #logical vector male dwarves
> names[male] #index names using logical vector males
> names[young & male] # young male dwarves
```

One important use of logical indexing is to categorize a continuous variable.

```r
> # simulate vector with 1000 age values
> age <- sample(0:100, 1000, replace = TRUE)
```
> mean(age); sd(age)
> agecat <- age # make copy
> # replace elements agecat with strings for q category
> agecat[age<15] <- "<15" # creating character vector
> agecat[age>=15 & age<25] <- "15-24"
> agecat[age>=25 & age<45] <- "25-44"
> agecat[age>=45 & age<65] <- "45-64"
> agecat[age>=65] <- "65+"
> table(agecat) # get freqs

2 indexing matrices and arrays

A vector has only one dimension, so it is indexed by a single number in a bracket. To index matrices and arrays, you have account for their additional dimensions.

2.1 indexing matrices

Create the following matrix.

> m<-matrix(round(rnorm(16,50,5)),2,2)
> dimnames(m)<-list(behavior=c("type A", "type B"), + MI=c("yes", "no"))
> m

Now do some indexing.

1. by position
   > m[1, ] # first row
   > m[1, , drop = FALSE]
   > m[1,2] # cell "d"

2. by name
   > m["type A",]
   > m[, "no"]

3. by logical
   > m[, 2] < 45 # logical vector
   > m[m[, 2] < 45] # data

You can achieve increasing levels of precision and complexity with indexing. In the following statement, (don’t submit it, it’s just for illustration) the extra comma after 3 tells R to return all the rows in x for which the 1st column is <3.

x[x[,1]<3,]
the extra comma after 3 tells R to return all the rows in x for which the 1st column is <3
The functions lower.tri() and upper.tri() use indexing to return the positions below or above a matrix.

```r
> m2 <- matrix(round(rnorm(81, 50, 5)), 3, 3)
> m2
> lower.tri(m2)
> upper.tri(m2)
> m2[lower.tri(m2)]
```

### 2.2 indexing arrays

Create the following array.

```r
> a <- array(sample(10:70, 8, rep=T), c(2, 2, 2))
> dimnames(a) <- list(exposure=c("e", "E"), disease=c("d", "D"),
                     + confounder=c("c", "C"))
> a
```

Now, index to return the cell count for unexposed, diseased, confounder negative individuals...

1. by position
   ```r
   > a[1, 2, 1]
   ```

2. by name
   ```r
   > a["e", "D", "c"]
   ```

3. by logical
   ```r
   > a[a==33]
   ```

### 3 indexing lists

Indexing lists can sometimes be challenging. Recall the bracket notation for lists, where double brackets refer to the "bin" of like objects, and a following single bracket refers to the contents of that bin.

```r
> l <- list(1:5, matrix(1:4, 2, 2),
            + c("John Snow", "William Farr"))
```

1. by position
   ```r
   > l[[1]]
   > l[[2]][2, 1]
   > l[[3]][2]
   ```
2. logical

```r
> char <- sapply(l, is.character)
> char
> epi.folk<-l[char]
> epi.folk
```

3.1 indexing the results of modeling

Indexing lists comes in handy when working with the results of statistical models, which frequently return results in the form of lists. Fortunately, most package authors return the results as named lists.

Work through the following conditional logistic regression of abortion and infertility to see an example of extracting list elements from the results of a model.

```r
> data(infert)
> library(survival) # package with clogit()
> mod1 <- clogit(case ~ spontaneous + induced +
+ strata(stratum), data = infert)
> mod1 # default results (7x risk c spont AB, 4x induced)
> str(mod1)
> names(mod1) #structure, names
> mod1$coeff # name to index result (list element)
> summod1<-summary(mod1) #more detailed results
> names(summod1) #detailed list components
```

4 indexing dataframes

Data frames can (generally) be indexed like matrices, with the added advantage of being able to use column (variable) names.

Run through this code to get a sense of how dataframes can be indexed.

data(infert)

1. position

```r
infert[1:4, 1:2]
infert[1:4, 2] <- c(NA, 45, NA, 23)
infert[1:4, 1:2]
```

2. name

```r
names(infert)
infert[1:4, c("education", "age")]
infert[1:4, c("age")] <- c(NA, 45, NA, 23)
```
infert[1:4, c("education", "age")]

3. logical

table(infert$parity)
# change values of 5 or 6 to missing
infert$parity[infert$parity==5 | infert$parity==6] <- NA

table(infert$parity)
table(infert$parity, exclude=NULL)

In the following perhaps more realistic example you will read in a set of anonymized hospital discharge data, and then index it in various ways.

> sparcs<-read.csv(file="http://www.columbia.edu/~cjd11/charles_dimaggio/DIRE/resources/R/sparcsShort.csv", stringsAsFactors=F)

• index rows

> brooklyn<-sparcs[sparcs$county=="59",]
> nyc<- sparcs$county=="58" | sparcs$county=="59" |
+ sparcs$county=="60" | sparcs$county=="61" | sparcs$county=="62"
> nyc.sparcs<-sparcs[nyc,]

• index columns

> dxs<-sparcs[,"pdx"]
> vars<-c("date", "pdx", "disp")
> my.vars<-sparcs[,vars]

• index rows and columns

> sparcs2<-sparcs[nyc,vars]

• variables to include

> brooklyn.sparcs<-subset(sparcs, county=="59",
+ select=c(date, pdx, disp))

• range of variables

> nyc.sparcs<-subset(sparcs, county=="59":"62",
+ select=c(county, pdx, disp))

• excluding rows

> nyc.sparcs<-subset(sparcs, county=="59":"62",
+ select=-c(county, pdx, disp))