MTH3240 R Notes 3

3 Vectors (Continued)

3.7 Comparison Operators

- R has several ***comparison operators*** that can be used on (scalar) variables as well as on vectors:

```
<                   # Less than
>                   # Greater than
==                  # Equal to
!=                  # Not equal to
<=                  # Less than or equal to
>=                  # Greater than or equal to
```

- Each one returns a `"logical"` value, `TRUE` or `FALSE`, depending on whether or not the relationship holds. Here are some examples:

```
5 < 6
```

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

```
5 == 6
```

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

```
5 != 6
```

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

- They can be used on `"character"` values too:

```
"a" == "b"
```

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

- When applied to vectors, they operate elementwise and return a `"logical"` vector:

```
x <- c(11, 3, 4, 12, 2)
y <- c(11, 9, 4, 12, 2)
```

```
x == y                                  # Compares x and y elementwise

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

- They can also be used to compare each element of a vector to a single value. For example (using `x` from above):

```
x > 10                                  # Compares each value in x to 10

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

- R coerces `TRUE` and `FALSE` to 1 and 0, respectively, when it needs to perform arithmetic on them. This is *very* useful, in combination with `sum()`, for *counting* how many values in a vector satisfy a given condition.

  To see, note that if we *sum* a bunch of 0's and 1's, it's the same as *counting* how many of them are 1's. Thus, to *count* how many elements of `x` are greater than 10, we can type:

```
sum(x > 10)                         # Counts how many elements of x are > 10

## [1] 2
```

  This shows that two of the values in `x` are greater than 10. (Above, `x > 10` is a `"logical"` vector.)

Section 3.1 Pre-Lab Exercises

Exercise 3.1 Consider the following vector:

```
x <- c(3, 4, 5)
```

Guess what the result of each of the following will be, then check your answers:

a) `x == 4`

b) `x > 4`

c) `x >= 4`

d) `x != 4`

Exercise 3.2 Consider the following two vectors:

```
x <- c(3, 4, 5)
y <- c(3, 4, 10)
```

Guess what the result of each of the following will be, then check your answers:

a) `x == y`

b) `x > y`

c) `x != y`

Exercise 3.3 Recall that R coerces `TRUE` and `FALSE` to 1 and 0, respectively, when it needs to. Guess what the result of each of the following will be, then check your answers:

a) `TRUE + TRUE + FALSE`

b) `sum(c(TRUE, TRUE, FALSE))`

Exercise 3.4 Consider the following vector:

```
x <- c(10, 8, -2, -6, -5)
```

Guess what will be returned by of each of the following commands, then check your answers:

a) `x > 0`

b) `sum(x > 0)`

Exercise 3.5 Consider the following two vectors:

```
x <- c(10, 8, 4, 7, 3)
y <- c(8, 8, 4, 9, 6)
```

Write a command involving `sum()` and `x == y` that counts how many of the elements of `x` are equal to their corresponding element of `y`. Make sure to check your answer.

3.8 Using `any()`, `all()`, and `which()`, `which.min()`, and `which.max()`

- The following are useful for searching vectors for values that satisfy a given condition:

```
any()        # Do any elements of a vector satisfy a given condition?
             # Returns TRUE or FALSE.
all()        # Do all of the elements of a vector satisfy a given
             # condition?  Returns TRUE or FALSE.
which()      # Returns the indices of the elements of a vector that
             # satisfy a given condition.
which.min()  # Returns the index of the minimum value in a vector.
which.max()  # Like which.min(), but returns the index of the maximum.
```

- `any()` tells us whether *any* values in a vector satisfy a certain condition:

```
x <- c(11, 3, 4, 12, 2)
any(x > 10)

## [1] TRUE
```

- `which()` determines *which* elements satisfy the condition, and returns their *indices*:

```
which(x > 10)

## [1] 1 4
```

  Thus we see that the 1st and 4th elements of `x` are greater than 10.

- `all()` tells us whether or not *all* of the values in a vector satisfy a condition:

```
all(x > 10)

## [1] FALSE
```

- We can use `any()`, `all()`, and `which()` for comparing *two* vectors. For example, consider the vectors:

```
x <- c(11, 3, 4, 12, 2)
y <- c(11, 9, 4, 12, 2)
```

  If we want to know which of the values in `x` are different from their corresponding value in `y`, we type:

```
which(x != y)

## [1] 2
```

We see that only the 2nd values of `x` and `y` differ.

- `which.min()` and `which.max()` return the *indices* of the smallest and largest values in a vector. For example

```
which.max(x)

## [1] 4
```

tells us that the largest value in `x` is the 4th value (12).

If multiple elements are tied for the smallest (or largest) value, `which.min()` (or `which.max()`) only returns the index of the one that occurs first in the vector:

```
z <- c(6, 3, 9, 3)
which.min(z)                    # Only returns the index of the first 3

## [1] 2
```

Section 3.2 Pre-Lab Exercises

Exercise 3.6 Consider the vector:

```
x <- c(2, 8, 6, 7, 1, 4, 9)
```

Guess what will be returned by of each of the following commands, then check your answers:

a) `any(x == 4)`

b) `all(x == 4)`

c) `which(x == 4)`

d) `which(x != 4)`

Exercise 3.7 Consider the vector:

```
x <- c(38, 22, 16, 14, 56)
```

a) Write a command that uses `any()` to determine if *any* of the values in `x` are greater than 30.

b) Write a command that uses `all()` to determine if *all* of the values in `x` are greater than 30.

c) Write a command that uses `which()` to determine *which* of the values in `x` are greater than 30.

Exercise 3.8 Consider the vectors:

```
x <- c(38, 22, 16, 14, 56)
y <- c(38, 21, 16, 41, 56)
```

a) Write a command to using `any()` and `x == y` determine if *any* of the values in `x` are equal to their corresponding value in `y`.

b) Write a command using `all()` to determine if *all* of the values in `x` are equal to their corresponding value in `y`.

c) Write a command using `which()` to determine *which* of the values in `x` are equal to their corresponding value in `y`.

Exercise 3.9 Consider the vector:

```
x <- c(38, 22, 16, 14, 56)
```

Guess what the result of each of the following commands will be, then check your answers:

a) `which.min(x)`

b) `which.max(x)`

3.9 Filtering

- ***Filtering*** refers to extracting from a vector those values for which some condition is met.

3.9.1 Extracting and Replacing Elements that Satisfy Some Condition

- To extract from a vector the elements that satisfy some condition, we just state the condition inside square brackets `[ ]`. For example:

```
x <- c(1, 3, 12, 5, 13)
x[x > 10]                          # Returns x "such that" x > 10.
                                   # Note that the expression x > 10 is
                                   # a "logical" vector
```

```
## [1] 12 13
```

Above, we extracted the elements of `x` that are greater than 10. Note that the expression `x > 10` is actually a `"logical"` vector:

```
x > 10

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

and the `TRUE`s indicate which elements of `x` are to be extracted.

- We can also use square brackets to *replace* elements that satisfy some condition. For example, to replace all of values in `x` that are greater than 10 by, say, 11, we can type:

```
x[x > 10] <- 11          # Replaces values that are greater than 10 by 11
x

## [1]  1  3 11  5 11
```

3.9.2 Using Values in One Vector to Extract Elements from Another

- Sometimes we need to extract from one vector the elements for which the values in *another* vector satisfy some condition. For example, suppose we have heights (inches) and weights (lbs) of 12 people:

```
ht <- c(69, 71, 67, 66, 72, 71, 61, 65, 73, 70, 68, 74)
wt <- c(175, 170, 210, 190, 195, 165, 163, 172, 158, 191, 213, 215)
```

To find the weights of the people who are taller than 72 inches, we type:

```
wt[ht > 72]              # Returns wt "such that" ht > 72.

## [1] 158 215
```

- This method is extremely useful with `"character"` vectors indicating group membership. For example, consider this data set:

Gender	Age
f	33
m	35
f	29
m	34
m	37
f	36
f	35
f	40
m	43
f	38
f	40
m	44

After creating the vectors:

```
Gender <- c("f", "m", "f", "m", "m", "f", "f", "f", "m", "f", "f", "m")
Age <- c(33, 35, 29, 34, 37, 36, 35, 40, 43, 38, 40, 44)
```

we can extract the ages of just the females by typing:

```
Age[Gender == "f"]          # Returns Age "such that" Gender == "f".

## [1] 33 29 36 35 40 38 40
```

Note that `Gender == "f"` is a `"logical"` vector:

```
Gender == "f"

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

Its `TRUE` values indicated which `Age`s to extract.

3.9.3 Extracting Elements that Satisfy Some Condition Using `subset()`

- Another way to extract from a vector the values that satisfy some condition is to use:

```
subset()          # Extract a subset of vector elements that satisfy a
                  # given condition
```

- `subset()` takes arguments `x`, a vector, and `subset`, a `"logical"` vector specifying the condition to be met by the values extracted from `x`. For example, to (again) extract from `Age` that correspond to females, we can type:

```
subset(x = Age, subset = Gender == "f")

## [1] 33 29 36 35 40 38 40
```

Section 3.3 Pre-Lab Exercises

Exercise 3.10 Consider again the vector:

```
x <- c(38, 22, 16, 14, 56)
```

Write a command involving square brackets `[ ]` and `x > 30` that extracts from `x` all the values that are greater than 30.

Exercise 3.11 Consider this data set, showing the genders, ages, and systolic blood pressures for 12 people:

Gender	Age	Blood Pressure
f	33	118
m	35	115
f	29	110
m	34	117
m	37	112
f	36	119
f	35	114
f	40	121
m	43	123
f	38	117
f	40	120
m	44	121

Here are the same data:

```
Gender <- c("f", "m", "f", "m", "m", "f", "f", "f", "m", "f", "f", "m")
Age <- c(33, 35, 29, 34, 37, 36, 35, 40, 43, 38, 40, 44)
BP <- c(118, 115, 110, 117, 112, 119, 114, 121, 123, 117, 120, 121)
```

a) Describe words what the following command does.

```
BP[Gender == "m"]
```

b) Describe words what the following command does.

```
BP[Age > 35]
```

3.10 `NA` Values

3.10.1 Introduction

- Data sets often contain ***missing values***, for example when a survey question was left unanswered or a laboratory measurement failed due to equipment malfunction.
- In R, missing values are represented by `NA`, which stands for "not available":

```
NA                    # Indicates a missing value ("not available")
```

- An `NA` value in a data set is like a "placeholder" for a data value that's supposed to be there but isn't:

```
x <- c(2.1, 4.1, NA, 4.4, 3.7)
x

## [1] 2.1 4.1  NA 4.4 3.7
```

- We can test for a missing value using `is.na()`:

```
is.na()                # Returns TRUE or FALSE depending on whether or
                       # not a value is NA
```

- When applied to a vector, `is.na()` returns a `"logical"` vector whose elements are `TRUE` or `FALSE` depending on whether the corresponding value in the original vector is `NA`:

```
x <- c(2.1, 4.1, NA, 4.4, 3.7)
is.na(x)

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

- To decide *which* values in a vector are missing, we can type:

```
which(is.na(x))

## [1] 3
```

3.10.2 Computing Summary Statistics from Data with `NA`s

- Most of the R functions for computing summary statistics return `NA` when passed a data set that contains `NA`s:

```
x <- c(2.1, 4.1, NA, 4.4, 3.7)
mean(x)                              # Returns NA because of the NA in x.

## [1] NA
```

Some of them have an optional argument `na.rm`, though, that can be used to remove the `NA`s before carrying out the calculations:

```
mean(x, na.rm = TRUE)       # Computes the mean of the non-NA values.

## [1] 3.575
```

3.10.3 Replacing `NA`s

- To replace the `NA`s in a vector by some value, like 0, we use:

```
x[is.na(x)] <- 0
x

## [1] 2.1 4.1 0.0 4.4 3.7
```

(Note that replacing `NA`s may not be appropriate.)

Section 3.4 Pre-Lab Exercises

Exercise 3.12 Consider the vector:

```
x <- c(1, 2, NA)
```

Guess what the result of each of the following will be, then check your answers.

a) `is.na(x)`

b)
```
x[is.na(x)] <- 0
x
```

Exercise 3.13 Consider the following vector:

```
x <- c(1, 2, NA)
```

a) Guess what the result of the following command will be, then check your answer:

```
mean(x)
```

b) The function `mean()` has an (optional) argument `na.rm`. If we specify `na.rm = TRUE`, any `NA`s are removed before the mean is computed. (Type `? mean` to see the help page.)

Guess what value the following command will return, then check your answer.

```
mean(x, na.rm = TRUE)
```