MTH 2520 R Notes 3

2 Vectors (Continued)

2.5 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

[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

[1]  TRUE FALSE FALSE  TRUE FALSE
```

- Recall that R coerces `TRUE` and `FALSE` to 1 and 0, respectively, when it needs to. This is *very* useful, in combination with the function `sum()`, when we want to count how many elements of a vector satisfy a given condition:

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

[1] 2
```

Section 2.5 Exercises

Exercise 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 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`

d) `> x != y`

Exercise 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 + FALSE + FALSE`

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

Exercise 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 5 Consider the following two vectors:

```
> x <- c(10, 8, -2, -6, -5)
> y <- c(8, 8, -2, -7, -5)
```

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

2.6 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 first one.

Section 2.6 Exercises

Exercise 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 7 Consider the vectors:

```
> x <- c(538, 432, 684, 716, 814, 624, 956)
> y <- c(538, 431, 648, 716, 841, 664, 656)
```

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

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

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

Exercise 8 Consider the vector:

```
> x <- c(538, 432, 684, 716, 814, 624, 956)
```

a) Write a command involving `any()` and `==` to determine if *any* of the values in `x` are divisible by 12. **Hint**: Use the remainder operator `%%`. The remainder will equal 0 if a value is divisible by 12.

b) Write a command involving `all()` and `==` to determine if *all* of the values in `x` are divisible by 12.

c) Write a command involving `which()` and `==` to determine *which* of the values in `x` are divisible by 12.

Exercise 9 Consider the vector:

```
> x <- c(538, 432, 684, 716, 814, 624, 956)
```

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

a) `> which.min(x)`

b) `> which.max(x)`

2.7 Computing Summary Statistics

- Several functions take vector arguments and compute summary statistics:

```
length()          # Number of elements in a vector
min(); max()      # Smallest and largest values in a vector
range()           # Range (smallest and largest values) of a vector
sum()             # The sum of the values in a vector
prod()            # The product of the values in a vector
cumsum()          # Cumulative sums of the values in a vector
cumprod()         # Cumulative products of the values in a vector
mean()            # The sample mean
median()          # Sample median
sd(); var()       # Sample standard deviation and variance
mad()             # Median absolute deviation
quantile()        # Sample quantile (percentile)
IQR()             # Interquartile range
summary()         # Five number summary (and sample mean)
```

- For a vector `x` containing the values $x_1, x_2, \ldots, x_n$, `mean(x)` computes the ***sample mean***, or arithmetic average, denoted $\bar{x}$:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- `median(x)` computes the ***sample median***, or ”middle value” after sorting the data:

$$\text{Median} = \begin{cases} \text{The middle sorted value if } n \text{ is odd.} \\ \text{The average of the two middle sorted values if } n \text{ is even.} \end{cases}$$

- `var(x)` computes the ***sample variance***, or ”average” squared deviation of an x_is away from the mean, denoted s^2:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- `sd(x)` computes the ***sample standard deviation***, which is the square root of the variance and represents a typical deviation of an x_i away from the mean:

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}.$$

Section 2.7 Exercises

Exercise 10 Consider the following data set:

```
> x <- c(10, 147, 7, 6, 7, 12, 9, 12, 11, 8)
```

a) Use `mean()` to compute the mean.

b) Use `median()` to compute the median.

c) Type

```
> sort(x)
```

Where does the median lie in the sorted data set?

d) Use `sd()` to compute the standard deviation of `x`

Exercise 11 The standard deviation measures variation in a set of data.

a) What do you think the standard deviation of the following data set will be? Check your answer using `sd()`.

```
[1] 5 5 5
```

b) Which of the following two data sets do you think will have a larger standard deviation? Check your answer using `sd()`.

```
[1] 5 6 7
```

```
[1]  1  6 11
```

2.8 Vectorized Computations

- We've seen that the operators '+', '-', '*', '/', and '^' operate elementwise on vectors.
- Many of R's built-in functions operate one element at a time too. For example, watch what happens when we pass a vector to `sqrt()`:

```
> x <- c(4, 9, 16, 25)
> sqrt(x)
```

```
[1] 2 3 4 5
```

- Functions that perform calculations on vectors one element at a time are said to be ***vectorized***.

Section 2.8 Exercises

Exercise 12 The function `abs()` takes the absolute value of a number. Guess what the result of the following command will be, then check your answer:

```
> x <- c(-1, 3, -4, -2)
> abs(x)
```

Exercise 13 Consider the following temperature measurements, in degrees Celsius:

```
> degreesC <- c(23, 19, 21, 22, 18, 20, 24, 25)
```

The relation between Celsius (°C) and Fahrenheit (°F) is:

$$^\circ\text{F} = \frac{9}{5} \cdot^\circ \text{C} + 32$$

In words, what will the following command do to the Celsius temperatures? Try it.

```
> (9/5) * degreesC + 32
```

Exercise 14 The following values represent radii of 8 circles:

```
> radii <- c(2, 2, 4, 5, 4, 3, 6, 7)
```

The area of a circle is related to its radius by:

$$\text{Area} = \pi \times \text{Radius}^2$$

where the number $\pi = 3.14159$ is `pi` in R. Write a command that creates a new vector named `areas` containing the areas of the 8 circles.

2.9 Filtering

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

2.9.1 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]                    # Note that 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.

- 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
  > x

  [1]  1  3 11  5 11
  ```

2.9.2 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 72 inches tall or taller, we type:

  ```
  > wt[ht >= 72]

  [1] 195 158 215
  ```

- 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"]

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

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

- Another way to extract from a vector the elements 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 elements extracted from `x`. For example, to (again) extract the elements from `x` that are greater than 10, we can type:

  ```
  > x <- c(1, 3, 12, 5, 13)
  > subset(x, subset = x > 10)

  [1] 12 13
  ```

Section 2.9 Exercises

Exercise 15 Consider again the vector:

```
>  x <- c(538, 432, 684, 716, 814, 624, 956)
```

a) Write a command involving square brackets `[ ]` that extracts from `x` all the values that are greater than 700.

b) Write a command involving square brackets `[ ]` that extracts from `x` all the values that are *not equal to* 814. **Hint**: Use the the comparison operator `!=`.

Exercise 16 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) After creating the three vectors, write a command that extracts the blood pressures of just the males. You may use either square brackets `[ ]` or the `subset()` function.

b) Write a command that extracts the ages of people whose blood pressures exceed 117. You may use either square brackets `[ ]` or the `subset()` function.

2.10 `NA` Values

2.10.1 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")
```

- 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
```

2.10.2 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)
```

```
[1] NA
```

- Many 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)

[1] 3.575
```

2.10.3 Extracting and Replacing `NA`s

- R treats `NA`s as *unknown* values. For example, below, `2 == NA` isn't `FALSE`, but `NA`, because R doesn't have enough information to determine what the value on the right side is:

```
> 2 == NA

[1] NA
```

  Likewise `NA == NA` isn't `TRUE` but `NA`:

```
> NA == NA

[1] NA
```

- For this reason we can't replace missing values by typing `x[x == NA] <- 0`. To replace the `NA`s by 0, we use `is.na()` as follows:

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

[1] 2.1 4.1 0.0 4.4 3.7
```

Section 2.10 Exercises

Exercise 17 Guess what the result of each of the following will be, then check your answers.

a) `> 3 == NA`

b) `> NA == NA`

Exercise 18 Consider the vector:

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

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

a) `> x == NA`

b)
```
> is.na(x)
```

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

Exercise 19 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) How can we compute the mean of a vector `x` that contains `NA`s, without having to remove the `NA`s from `x` first? **Hint**: Look at the help file for `mean()` by typing

```
> ? mean
```

In particular, look in the help file for the argument `na.rm`.