Vectors

- Vectors may be created using the `c()` function. Separate vector elements by commas.

```
> a <- c(1, 7, 32, 16) # Again, spaces have no effect, but improve
      readability.
> a
```

```
[1]  1 7 32 16
```

- Vectors do not need to consist of numbers; vectors of character data or logicals are allowed, too:

```
# A vector of character strings.
> wind <- c("north", "west", "south", "east")
> wind
```

```
[1] "north" "west"  "south" "east"
```

```
> logic <- c(TRUE, FALSE, TRUE) # A vector of truth values.
> logic
```

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

- Sequences of integers may be created using a colon (`:`).

```
> b <- 1:10
> b
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

```
> c <- 20:15
> c
```

```
[1] 20 19 18 17 16 15
```

- Other regular vectors may be created using the `seq()` (sequence) and `rep()` (repeat) commands.

```
> d <- seq(1, 5, by = 0.5)
> d
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
> e <- seq(0, 10, length = 5)
> e
```

```
[1]  0.0  2.5  5.0  7.5 10.0
```

```
> f <- rep(0, 5)
> f
```

```
[1] 0 0 0 0 0
```

```
> g <- rep(1:3, 4)
> g
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

```
> h <- rep(4:6, 1:3) # Study what this does!
> h
```

```
[1] 4 5 5 6 6 6
```

- Vectors of random numbers can be created with a set of functions that start with `r`, such as `rnorm()` (draw numbers from a normal distribution) or `runif()` (draw numbers from a uniform distribution in the interval $(0, 1)$).

```
> x <- rnorm(5) # Standard−normal random numbers
> x
```

```
[1] -1.4086632  0.3085322  0.3081487  0.2317044 -0.6424644
```

```
> y <- rnorm(7, 10, 3) # Normal random numbers with mu = 10, sigma = 3
> y
```

```
[1] 10.407509 13.000935  8.438786  8.892890 12.022136  9.817101  9.330355
```

```
> z <- runif(10) # Uniform random variables in the interval (0, 1)
> z
```

```
[1] 0.925665659 0.786650785 0.417698083 0.619715904 0.768478685 0.676038428
[7] 0.050055548 0.727041628 0.008758944 0.956625536
```

- If a vector is passed to an arithmetic calculation, it will be applied element-by-element.

```
> c(1, 2, 3) + c(4, 5, 6)
```

```
[1] 5 7 9
```

If the vectors involved are of different lengths, the shorter one will be repeated until it has the same length as the longer one.

```
> c(1, 2, 3, 4) + c(10, 20)
```

```
[1] 11 22 13 24
```

```
> c(1, 2, 3) + c(10, 20)
```

```
[1] 11 22 13
Warning message:
longer object length is not a multiple of shorter object length in: c(1, 2,
    3) + c(10, 20)
```

- Basic mathematical functions will apply element-by-element as well.

```
> sqrt(c(100, 225, 400))
```

```
[1] 10 15 20
```

- To select subsets of a vector, use square brackets (`[...]`). Inside the square brackets, you write (or construct) a vector that specifies which elements you wish to select.

```
> d
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
> d[3] # Third element of vector d
```

```
[1] 2
```

```
> d[5:7] # Elements 5 to 7
```

```
[1] 3.0 3.5 4.0
```

```
> my.selection = 5:7 # Create a vector
> d[my.selection] # Use it to select elements
```

```
[1] 3.0 3.5 4.0
```

- If you put a logical vector inside the brackets, R will return only those elements of `d` where the logical vector has value `TRUE`. For instance:

```
> d[c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE, TRUE)]
```

```
[1] 3.0 3.5 4.0 4.5 5.0
```

So, the first four values of `d` are not selected, because the first four values of the logical vector are `FALSE`. The last five values *are* selected, because the last five values of the logical vector are `TRUE`.

This technique can be very useful because it is very easy to generate useful logical vectors. Here's an example:

```
> d > 2.8 # Which elements of d are larger than 2.8?
```

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

Note that this creates a logical vector that specifies which elements of `d` are larger than 2.8.

Here's another example:

```
> d == 3 # Which elements of d are equal to 3?
```

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

Using this trick, we can easily select all elements of `d` that are greater than 2.8:

```
> d[d > 2.8]
```

```
[1] 3.0 3.5 4.0 4.5 5.0
```

Similarly, you can use this trick to select a specific range of the data:

```
> d[d > 2 & d < 4] # Select numbers between 2 and 4
```

```
[1] 2.5 3.0 3.5
```

This (not so useful) code select the value 3:

```
> d[d == 3]
```

```
[1] 3.0
```

This (very useful) code selects the elements of `d` that are *not* equal to 3:

```
> d[d != 3]
```

```
[1] 1.0 1.5 2.0 2.5 3.5 4.0 4.5 5.0
```

(In other words, this removes all elements with the value 3.0 from the vector!)

Make sure you understand this technique; it is very powerful!

- The number of elements in a vector can be found with the `length()` function.

  ```
  > length(d)
  [1] 9
  ```

  What is being calculated in the following command?

  ```
  > length(d[d > 2.8])
  [1] 5
  ```

˜ **Problem 1.** Create the following vectors in R.

$$a = (5, 10, 15, 20, ..., 160)$$
$$b = (87, 86, 85, ..., 56)$$

Use vector arithmetic to multiply these vectors (element by element) and call the result *d*. Select subsets of *d* to identify the following.

a) What are the 19th, 20th, and 21st elements of *d*?

b) List all the elements of *d* which are less than 2000?

c) How many elements of *d* are greater than 6000?

˜ **Problem 2.** Create a character vector named char that has capital letters ”A”, "B", "C", "D" repeated four times (Hint: use function LETTERS)

˜ **Problem 3.** Create a numerical vector named numbers using the runif() function consisting of 30 values between -2 and 10. Read help function on runif first.

a) Select all values greater than 3. How many do you get? (Hint: use function-length)

b) What is the sum of the values greater than 3?

c) Re-create another vector using the same code. How many values are now greater than 3?