# container_types

September 23, 2020

Container Types in R
Paul Stey, Ph.D.
2020-09-13

## 1 1. Container Types

- What is a container type?
- Examples:
  - `vector`
  - `matrix`
  - `array`
  - `list`

## 2 2. The `vector` Type

The `vector` type in R is a one-dimensional array that can hold homogeneous data (i.e., data that is all of the same type).

```
In [ ]: my_vec <- c(42, 137, 3)              # use c() function to combine elements in to vecto

        print(my_vec)                        # Note: all elements are of type integer
```

```
In [ ]: animal_vec <- c("dog", "cat", "bird", "potato")

        print(animal_vec)
```

### 2.1 2.1 Accessing Elements (i.e., "indexing")

- Once we have a vector, we need to be able to access its elements
- This is done using the `[]` notation.

```
In [ ]: people <- c("ned", "jon", "robb", "arya")
```

```
In [ ]: people[3]                  # use [] notation to access element
```

### 2.1.1 2.1.1 Range-Based Indexing

- We often want more than a single element from a vector
- Range-based intexing is one method of getting "slices" of a vector (i.e., contiguous elements)

```
In [ ]: print(people)                    # Use vector from previous example

        people[3:4]                     # use range notation to get "slice"
```

## 2.2 2.2 Indexing with Vector

- Suppose we want a non-contiguous set of elements
- We can use a vector of integers for indexing

```
In [ ]: vec_idx <- c(1, 4)

        people[vec_idx]
```

```
In [ ]: people[c(1, 3)]
```

## 2.3 2.3 Indexing with Booleans

- We are also able to use boolean values to index

```
In [ ]: print(people)
```

```
In [ ]: is_stark <- c(TRUE, FALSE, TRUE, TRUE)

        people[is_stark]
```

## 2.4 2.4 Modifying Elements

- We can use indexing or slicing to modify a vector's elements

```
In [ ]: animal_vec <- c("dog", "cat", "bird", "potato")

        print(animal_vec)
```

```
In [ ]: animal_vec[2:4] <- "fish"          # replace 4th elements

        print(animal_vec)                 # potato is not an animal
```

# 3 3. The `matrix` Type

- 2-dimensional array (i.e., height and width)
- Store homogeneous data (i.e., all same type)

```
In [ ]: dat <- 1:12                  # generate sequence from 1 to 12

        m <- matrix(dat, nrow = 3)  # put `dat` in to a matrix
```

```
In [ ]: print(m)
```

## 3.1 3.1 Non-Numeric Data

- Like the `vector` type, we can also store non-numeric data in a `matrix`

```
In [ ]: some_names <- c("lee", "sue", "bill", "don")

        name_mat <- matrix(some_names, nrow = 2)
```

```
In [ ]: print(name_mat)
```

## 3.2 3.2 Indexing `matrix` Object

- Indexing and slicing work much like in `vector` objects

```
In [ ]: x <- matrix(rnorm(8), nrow = 4)

        print(x)
```

```
In [ ]: x[1, 2]                    # element in first row, second column
```

```
In [ ]: x[3, 2]                    # third row, second column
```

### 3.2.1 3.2.1 Getting a "slice"

- Can also using indexing to get an entire row or column

```
In [ ]: w <- matrix(1:12, nrow = 4)

        print(w)
```

```
In [ ]: w[2, ]                     # get second row
```

```
In [ ]: w[, 1]                     # get first column
```

### 3.2.2 3.2.2 Matrix Slicing (cont.)

```
In [ ]: row_idx <- c(2, 4)
        col_idx <- 3

        w[row_idx, col_idx]
```