Vectors in R

CSC121
Mark Kazakevich
Let’s talk about...

> 2 + 5

[1] 7

This thing
Algorithm + Data + Structure = Program

- So far, we’ve seen a few examples these different parts of a program
Data Structures

Algorithm + Data + Structure = Program

- We know that all programming languages have to handle data.
- Every programming language provides ways to organize and store data to make it easier and efficient to work with.
- These are called **Data Structures**.
- We’re going to learn about one of R’s fundamental data structures that helps us store and work with data more easily.
Vectors

- All the numeric values we’ve seen so far are stored in an R data structure called a ‘Vector’

- A vector in R is an ordered group of values
  - Every value in a vector is called an ‘element’ of that vector
Vectors with one element

- All of the numbers we’ve been using are really one-element vectors!
  - For example, R automatically creates a vector every time we type a number into the Console and press Enter
  - For example...
Examples of Vectors

> 6
[1] 6

A one-element vector with the number ‘6’ in it

This ‘1’ means that the value directly beside it is the first element in the vector, in this case, 6

The ‘1’ does NOT mean there is only one element in the vector
Vectors with more than one element

- R’s vector data structure allows us to group many elements together

- To quickly make a multi-element vector, R provides a function:
  
  \(c(\ldots)\)

  - ‘c’ stands for ‘combine’.
  - This function combines its arguments into a vector.
  - The multiple dots ‘…’ means multiple arguments with no specific limit
Creating a multi-element vector

A vector with two elements

> c(4, 7)

[1] 4 7

A two-element vector with the numbers 4 and 7.

• Notice how it still only says ‘1’. This is because the value directly beside it is the first element of the vector, 4. 7 is the second element of the vector.

• Let’s talk more about the order of these elements...
Elements have an ordering

- The elements in a vector are ordered.
- When creating a vector with the `c(...)` function, the elements take the order that the arguments to the function were in.

```r
> c(23, 9, 534)  # The order of the elements is:
[1] 23 9 534
First element: 23
Second element: 9
Third element: 534
```
Longer vectors

- Vectors can be quite long
  - Maximum size is about 2 billion elements
  - Don’t try this though...your computer will likely run out of memory and crash!
Longer Vectors

- Often, a large vector printed to the console is too long for one line
- This is where our [1] comes in

\[
\begin{array}{c}
\text{c}(4, 7, 534, 938, 6432, 535) \\
[1] \ 4 \ 7 \ 534 \ 938 \ 6432 \\
[6] \ 535
\end{array}
\]

- The element to the right of the [1] is the first element of the vector
- The element to the right of the [6] is the sixth element of the vector
Working with Vectors

- Just like any other data in R, vectors can be stored in a variable

```r
> v <- c(24, 5, 347)
> v
[1] 24 5 347
```

- Vectors can be passed as arguments to functions:

```
FunctionWithVectorArgument(v)
```
Indexing

● Often you’ll be working with a vector, and you’ll want to isolate one of its elements

● **Indexing**
  ○ The *position* of each element in a vector is called its index
    ■ First element: index 1
    ■ Second element: index 2
    ■ Third element: index 3
    ■ etc..
Indexing

\[ \texttt{c(24, 5, 347)} \]

\[ \texttt{[1] 24 5 347} \]

- If we have the vector above, we say:
  - The element at index 1 is 24
  - The element at index 2 is 5
  - The element at index 3 is 347

- Order matters! \( \texttt{c(5, 7)} \) is not the same as \( \texttt{c(7, 5)} \)
Indexing

> v <- c(24, 5, 347)

- We can access the element at each index using **vector indexing**:

  > v[1]
  > [1] 24

  > v[2]
  > [1] 5

  > v[3]
  > [1] 347

- Notice how we get a new value in its own one-element vector, which is what we’re used to seeing
Indexing multiple elements

- We can access multiple values to create new vectors from the original vector.
- We can use the colon `:` to get a range of elements from the vector.
- In general, given a vector $v$, and two positive integers $x$ and $y$, $v[x:y]$ gives a vector with the elements of $v$ from index $x$ to index $y$ (inclusive).
Indexing multiple elements

> v <- c(24, 5, 347, 97, 43)

- Given the vector above
  - We can use the colon `:` to get a range of elements from the vector. Think of it as getting a ‘slice’ of the vector.

> v[1:3]
[1] 24 5 347

> v[4:5]
[1] 97 43
Can find length of vectors

```r
> v <- c(24, 5, 347, 97, 43)
> length(v)
[1] 5

> v[length(v)]
[1] 43
```
Type of a Vector

- The type of the vector is defined by the type of its elements
- All elements in a vector must be of the same type
- If you put different types in, R will find a way to convert them to force them to be the same type

```r
> v <- c(24, 5, 347)
> typeof(v)
[1] "double"
```
Type of a Vector

- The type of the vector is defined by the type of its elements.
- All elements in a vector must be of the same type.
- If you put different types in, R will find a way to convert them to force them to be the same type.

```r
> v <- c(24, as.integer(5), 347)
> typeof(v)
[1] "double"
```

R converts the integer value to a double to ensure all values have the same type.
Let’s look at vectors in RStudio
Let’s write a function for finding the distance between two points on a graph.

You are given the two points \((-1, 2)\) and \((2, 6)\), and you want to find the distance between them.

The points look like this:

Distance formula

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$