Working with vectors and data types in R

Learning Objectives

At the end of this activity, you will be able to:

- Understand the structure of and be able to create a vector object in R.

What you need

You need R and RStudio to complete this tutorial. Also we recommend have you have an `earth-analytics` directory setup on your computer with a `/data` directory with it.

- How to Setup R / R Studio
- Setup your working directory

Vectors and data types

A vector is the most common data structure in `R`. A vector is defined as a group of values, which most often are either numbers or characters. You can assign this list of values to an object or variable, just like you can for a single value. For example we can create a vector of animal weights:

```
weight_g <- c(50, 60, 65, 82)
weight_g
## [1] 50 60 65 82
```

A vector can also contain characters:

```
animals <- c("mouse", "rat", "dog")
animals
## [1] "mouse" "rat"   "dog"
```

There are many functions that allow you to inspect the content of a vector. `length()` tells you how many elements are in a particular vector:

```
length(weight_g)
## [1] 4
length(animals)
## [1] 3
```

Vector data types

An important feature of a vector, is that all of the elements are the same data type. The function `class()` shows us the class (the data type) of an object:

```
class(weight_g)
## [1] "numeric"
class(animals)
## [1] "character"
```

The function `str()` shows us the **structure** of the object and the elements it contains. `str()` is a really useful function when working with large and complex objects:

```
str(weight_g)
##  num [1:4] 50 60 65 82
str(animals)
##  chr [1:3] "mouse" "rat" "dog"
```

You can add elements to your vector by using the `c()` function:

```
# add the number 90 to the end of the vector
weight_g <- c(weight_g, 90)

# add the number 30 to the beginning of the vector
weight_g <- c(30, weight_g)
weight_g
## [1] 30 50 60 65 82 90
```

In the examples above, we saw 2 of the 6 **atomic vector** types that `R` uses:

1. `"character"` and
2. `"numeric"`.

These are the basic data tpes that all `R` objects are built from. The other 4 are:

- `"logical"` for `TRUE` and `FALSE` (the boolean data type)
- `"integer"` for integer numbers (e.g., `2L`, the `L` indicates to R that it's an integer)
- `"complex"` to represent complex numbers with real and imaginary parts (e.g., `1+4i`) and that's all we're going to say about them
- `"raw"` that we won't discuss further

Data type vs. data structure

Vectors are one of the many **data structures** that `R` uses. Other important ones include: lists (`list`), matrices (`matrix`), data frames (`data.frame`) and factors (`factor`). We will look at `data.frames` when we open our `boulder_precip` data in the next lesson!

Optional challenge activity

- **Question**: What happens when we create a vector that contains both numbers and character values? Give it a try and write down the answer.
- **Question**: What will happen in each of these examples? (hint: use `class()` to check the data type of your objects):

```
num_char <- c(1, 2, 3, 'a')
num_logical <- c(1, 2, 3, '2.45')
char_logical <- c('a', 'b', 'c', frog)
tricky <- c(1, 2, 3, '4')
```

- **Question**: Why do you think it happens?
- **Question**: Can you draw a diagram that represents the hierarchy of the data types?

Subsetting vectors

If we want to extract one or several values from a vector, we must provide one or several indices in square brackets. For instance:

```
animals <- c("mouse", "rat", "dog", "cat")
animals[2]
## [1] "rat"
animals[c(3, 2)]
## [1] "dog" "rat"
```

Data Tip: R indexes start at 1. Programming languages like Fortran, MATLAB, and R start counting at 1, because that's what human beings typically do. Languages in the C family (including C++, Java, Perl, and Python) count from 0 because that's simpler for computers to do. {: .notice }

Subset Vectors

We can subset vectors too. For instance, if you wanted to select only the values above 50:

```
weight_g > 50    # will return logicals with TRUE for the indices that meet the condition
## [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## so we can use this to select only the values above 50
weight_g[weight_g > 50]
## [1] 60 65 82 90
```

You can combine multiple tests using `&` (both conditions are true, AND) or `|` (at least one of the conditions is true, OR):

```
weight_g[weight_g < 30 | weight_g > 50]
## [1] 60 65 82 90
weight_g[weight_g >= 30 & weight_g == 21]
## numeric(0)
```

When working with vectors of characters, if you are trying to combine many conditions it can become tedious to type. The function `%in%` allows you to test if a value is found in a vector:

```
animals <- c("mouse", "rat", "dog", "cat")
animals[animals == "cat" | animals == "rat"] # returns both rat and cat
## [1] "rat" "cat"
animals %in% c("rat", "cat", "dog", "duck")
## [1] FALSE  TRUE  TRUE  TRUE
animals[animals %in% c("rat", "cat", "dog", "duck")]
## [1] "rat" "dog" "cat"
```

Optional challenge

- Can you figure out why `"four" > "five"` returns `TRUE`?