Week 2: Classes in R

James Robertson

2025-01-13

What are guided walkthroughs?

Guided walkthroughs serve as a reference and review for the week’s material. A walkthrough will be available for every week starting with Week 2. These walkthroughs will go through the material covered in the videos, offering additional worked examples and debugging suggestions.

This document will feature text, coding examples, and corresponding output (see below)

```r
fact<-'Real R code!'
#With a comment!
print('And its output looks like this!')
```

```
## [1] "And its output looks like this!"
```

Reading the assigned walkthroughs will help best ensure your success in the course. You are encouraged to re-reference a week’s walkthrough whenever you encounter errors or the material is confusing! If you find the topics explored here fascinating, please come to my office hours so we can discuss topics in *even greater detail*.

Objects and Classes

Functions covered:

- `typeof()`
  - Provides the type of an object. What that “type” represents is a bit technical, mostly about
- `str()`
  - Provides the structure of an object, meaning it will show the object’s value as well as its **Class**
- `list()`
  - Joins objects into a **List**
- `c()`
  - *Combines* objects to form a **Vector**

Definitions

- **Objects** are `data structures with attributes`

    - In *plain English*, **Objects** are *things* that take *values*. What kind of values they can take and how they are stored are based on the **Class** of **Object** (see below).
- **Methods** are `procedures which act on objects based on attributes`
    - Not very clear! In short, **Methods** are ways you can manipulate **Objects**. How exactly **Methods** act upon a given **Object** is based on the **Object**'s **Class**.
- **Classes** are the types of **Objects** and determine how the **Objects**' values are saved.
    - No new technical terms in this definition, "types" are just what you think, "saved" as in "saved to your computer".

Specific **Classes** will be discussed in detail next week, but this week we will give a brief overview of how **Classes** show up in `R`.

Examples

There is not a lot of content this week code-wise, mostly getting a grasp of RStudio and learning your way around. Since we are interested this week in **Classes** and **Objects**, let's look at some examples.

Here's a quick example of checking the type of an object. Below I create the `myList` object using the `list()` function. As discussed in the lectures, you can generally tell what **Class** an object is from the function used to create it.

```
myList<-list('A list','With two elements!')
typeof(myList)
```

```
## [1] "list"
```

```
str(myList)
```

```
## List of 2
##  $ : chr "A list"
##  $ : chr "With two elements!"
```

Wow, an object made with `list()` is a **List**! Astonishing! Using `str()` then gets us more information, too, telling us what its values look like. What else can we look at?

Let's look next at the `c()` function. The `c()` gets its name from "combine" or "concatenate" (depending on who you ask), it's a function for combining things to form a **Vector**. Though everything created with `c()` is a vector, there can be some big differences between them! Let's have a look at the below.

```
myCVector<-c('A vector','With two elements!')
typeof(myCVector)
```

```
## [1] "character"
```

```
str(myCVector)
```

```
##  chr [1:2] "A vector" "With two elements!"
```

```
myNVector<-c(1,10,20)
typeof(myNVector)
```

```
## [1] "double"
```

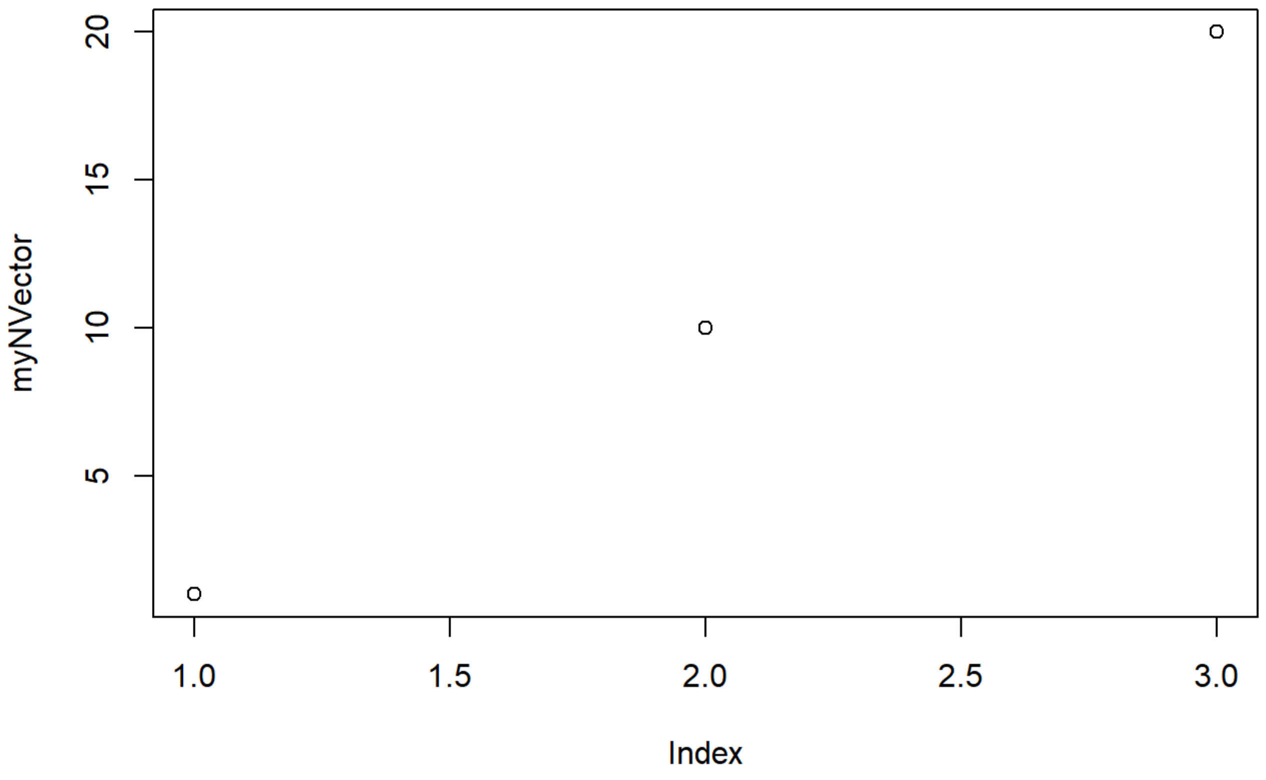
```
str(myNVector)
```

```
##  num [1:3] 1 10 20
```

Why do they have different types if they were *both* made with `c()` ? First of all, The `c()` function creates a **Vector**, but what *type* of vector depends on its values. If the values are numbers, it's *numeric* (but type `double` (code-speak for "number")). If it's text (called **Strings**) then it's *character*.

How do these differences in Objects show up when using functions on them?

```
plot(myNVector)
```



By default `plot()` will make a scatterplot when given a numeric vector. On the Y-axis are the values of the vector (1, 10, 20) and on the X-axis are their positions in the vector (1st, 2nd, 3rd). What about our character vector?

```
plot(myCVector)
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): NAs introduced by coercion
```

```
## Warning in min(x): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x): no non-missing arguments to max; returning -Inf
```

```
## Error in plot.window(...): need finite 'ylim' values
```

Errors! And Warnings! And a giant empty space! So `plot()` doesn't like being given character vectors. These messages are kind of confusing, but the real important one is the first Warning: `NAs introduced by coercion`. This means that `R` tried to turn our character vector into numbers (coercion) and it could not find a sensible way to do that (big shock!). Thus the values turned into `NA` (Not Applicable, *no value*!) and then `R` couldn't figure out how to plot non-values. Thus we get a big blank space where a plot might go and a bunch of other complaints spiraling out of those `NA` values.

There are a few different Classes of Objects in `R` (and a seemingly unlimited number of functions), but **List** and **Vector** are some of the most basic and most commonly used. I don't want to get too ahead of myself, though, so the technicalities of these and other Classes will have to wait until next week! In the meantime, remember to ask your instructor *lots of questions*!