# R programming: vectors

2024-09-02

- **M1 MIDS**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

Vectors in R

`vctrs` package

**Objectives**

## Atomic vectors

In `R` parlance, *vectors* denote very general forms of sequences, that is objects that can be indexed using `[[]]`, subseted/sliced using `[]`, and combined using `c()`. We often confuse vectors and *atomic vectors*. Figure 1 from Advanced R by Wickham outlines that *atomic vectors* are special cases of vectors (just as *lists*).

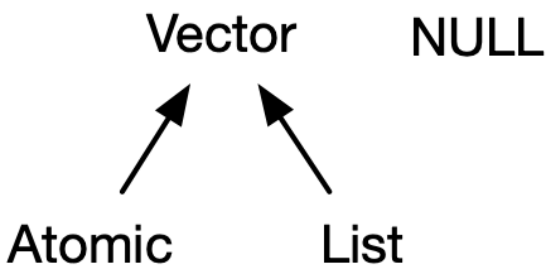

Figure 1: Atomic vectors as vectors

**Reading assignments**

Read first Vectors in R for Data Science 1st editon, then Vectors in Advanced R programming.

In words, atomic vectors are homogeneous vectors where all items have the same type. This criterion is questionable, since defining the type of an object in `R` is not obvious. There is a type hierarchy, and objects may have several types. Nevertheless we may use `typeof()` to determine the *storage* mode of an object.

## Basic atomic vectors

Basic atomic vectors are sequences of objects with the simplest storage modes.

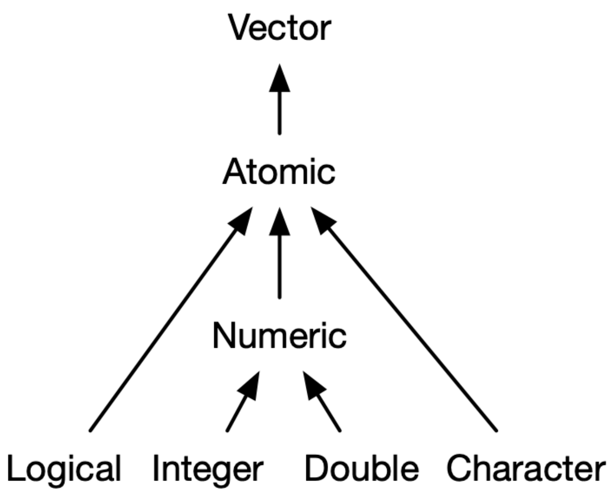

Figure 2: Common atomic vectors

## Null values

**Question**

Try to determine which items in a vector are NULL

```
x <- c(NA, 3, NA, 7, 13)
x == NA
```

Explain the output. Fix it.

**Question**

What happens when you combine (with `c()`) atomic vectors with different base types?

```r
x <- c(1L:3L)
y <- letters[5:9]
z <- rep(c(TRUE, FALSE), 2)[1:3]
x ; y ; c(x,y) ;  c(x,z) ; c(y,z)
```

## Attributes

**Question**

Attributes are metadata.

```r
x <- as_date("2024-08-06") + 1:7
is_vector(x) ; is_atomic(x) ; class(x) ; typeof(x)
```

```
[1] TRUE
[1] TRUE
[1] "Date"
[1] "double"
```

```r
attributes(x)
```

```
$class
[1] "Date"
```

```r
names(x) <- wday(x, label=T, abbr=F)

x
```

```
    mercredi        jeudi     vendredi       samedi     dimanche        lundi
"2024-08-07" "2024-08-08" "2024-08-09" "2024-08-10" "2024-08-11" "2024-08-12"
       mardi
"2024-08-13"
```

```r
attributes(x)
```

```
$class
[1] "Date"

$names
[1] "mercredi" "jeudi"    "vendredi" "samedi"   "dimanche" "lundi"    "mardi"
```

```r
x[["mercredi"]]
```

```
[1] "2024-08-07"
```

```r
attr(x, "names")
```

```
[1] "mercredi" "jeudi"    "vendredi" "samedi"   "dimanche" "lundi"    "mardi"
```

## Less basic atomic vectors

**Question**

What are `raw` vectors good for?

**Question**

What is the difference between `POSIXlt` and `POSIXct`? (Ask chatgpt)

**Question**

What does `is.atomic()` do?

**Question**

Is it possible to have an atomic vector with type/class `POSIXct`? `POSIXlt`? Are the answers of `class`and `typeof` always identical/consistent?

**Question**

Explain the following

```r
x <- "A Man A Plan a Canal Panama"
y <-  rep("A Man A Plan a Canal Panama", 5)
is.character(y) ; obj_size(x) ; obj_size(y)
```

```
[1] TRUE
136 B
176 B
```

**Question**

Is an object of type `factor` a vector? an atomic vector?

**Question (Exercise 20.4.6.2 from R for Data Science 1st Ed)**

Carefully read the documentation of `is.vector()`. What does it actually test for? Why does is.`atomic()` not agree with the definition of atomic vectors above?

Factors, Dates, and Date-times are cases of Augmented vectors.

## Recalling S3 classes

A basetype object with at least a `class` attribute.

Attribute `Class` is used to implement the S3 object oriented system.

In the next chunk `x` is a vector with basettype `double` but class `Date`. Each item in `x` is interpreted as the number of days ellapsed since the origin of time according to Unix `1970-01-01`. It is printed accordingly.

```
x <- as.Date("2024-08-06") + 1:7

class(x) ; typeof(x)
```

```
[1] "Date"
```

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

```
#| code-fold: false
x |>
  unclass()|>
  str()
```

```
 num [1:7] 19942 19943 19944 19945 19946 ...
```

`as.Date()` is an example of generic function from base `R`.

```
as.Date
```

```
function (x, ...)
UseMethod("as.Date")
<bytecode: 0x63f723551c10>
<environment: namespace:base>
```

The chunk above used method `as.Date.character()`

```
methods("as.Date")  # methods("as.Date")
```

```
[1] as.Date.character   as.Date.default     as.Date.factor
[4] as.Date.numeric     as.Date.POSIXct     as.Date.POSIXlt
[7] as.Date.vctrs_sclr* as.Date.vctrs_vctr*
see '?methods' for accessing help and source code
```

```
getS3method("as.Date", "character")  # as.Date.character
```

### Examples

Factors have basetype `integer` and attribute `factor` and `levels`.

```
ctr_names <- factor(ISOcodes::ISO_3166_1$Name)

ctr_names |>
  str()
```

```
 Factor w/ 249 levels "Afghanistan",..: 13 1 7 8 2 3 6 234 11 12 ...
```

```
class(ctr_names); str(attributes(ctr_names))
```

```
[1] "factor"
```

```
List of 2
 $ levels: chr [1:249] "Afghanistan" "Åland Islands" "Albania" "Algeria" ...
 $ class : chr "factor"
```

```
ctr_names |>
  unclass()  |>
  str()
```

```
 int [1:249] 13 1 7 8 2 3 6 234 11 12 ...
 - attr(*, "levels")= chr [1:249] "Afghanistan" "Åland Islands" "Albania" "Algeria" ...
```

Examples of important S3 classes

- `lm`
- `kmeans`
- `prcomp`
- `hclust`

**Question**

Explain

```
ctr_names |>
  str_to_upper() |>
  str()
```

```
 chr [1:249] "ARUBA" "AFGHANISTAN" "ANGOLA" "ANGUILLA" "ÅLAND ISLANDS" ...
```

## History

### Relevance: S3 generics

An S3 object behaves differently from its underlying base type when it is passed to a *generic* function.

What is a *generic*?

**Question**

- What happens if an S3 object is passed to a *generic*?
- What is method dispatch?
- What kind of MRO (Method Resolution Order) is used?
- How would you register a new method for a generic?
- How do you define a generic?
- Give examples of generics in base `R`.
- Get the list of base `R` functions which are generics.

**Question**

#### Preserving attributes

## S3 vectors as collections

In **?@lst-simple-loop**, `collection` may denote a list, a vector, or any other iterable sequence you can encounter in `R`.

Vectors deserve special consideration.

Documentation

S3 vectors in Advanced R Programming

## Desirable properties of vectors

### Combining vectors using `c()`

## `vctrs` and S3 vectors

Package `vctrs` makes the life of developpers who rely on S3 vectors easier.

### Creating a new S3 vector class

In package `nycflights13`, in tibble `flights`, columns with names ending with `dep_time` or `arr_time` have basetype `integer`.

```
stopifnot(
  require(nycflights13)
)
```

```
Loading required package: nycflights13
```

```
flights |>
  select(ends_with('_time')) |>
  glimpse()
```

```
Rows: 336,776
Columns: 5
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
```

Nevertheless, these columns encode time information (hour, minute, second) in an unusual way. The last two digits represent minutes, the leading digits represent hours. In the sequel, we define an S3 vector class with basetype `integer` that will allow us to handle these columns in a transparent way. Desirable properties are

- Readable display: `517` should be displayed as `5h17m`

- Some time arithmetics should be possible: we should be able either to add `difftime` or to compute the difference between `dep_time` and `sched_dep_time`
- Some validation should be possible: `2517` is not a valid value for `dep_time`
- Casting to `datetime` should be easy
- Casting from `datetime` should be easy as well
- ...

We use the tools from article S3 vectors

**Question**

Create a new S3 vector class called `weird_tm`. Endow it with a constructor `new_weird_tm()`, an helper `weird_tm()`, a test `is_weird_tm()`.

**Question**

Define a `format()` function for class `weird_tm`. Mind NAs.

**Question**

Casting and coercion
The next piece of code does not work

```
c(weird_tm(flights$dep_time[1:5]), flights$dep_time[1:5])
```

We need to define casting methods for generics `vec_cast()` and `vec_ptype2()` at least for casting to `integer` and `character`.

**Question**

Transform the tibble `flights` so that columns with name ending with `_time` (except `air_time`) have type `weird_time`. Is it still possible to filter rows with `dep_time` is a prescribed time interval.

We will use tools from `vctrs` to define differences between `weird_tm` objects.

**Question**

Double dispatch.

**Question**

Define the difference `-` operator for two vectors of class `weird_tm`. The result is expected to be an `integer` vector.

## Further reading

- Object Oriented Programming in R part 1 to ...

https://www.youtube.com/watch?v=P3FxCvSueag