Data types in R

ANNETTE MUTHONDU/C029/404217/2024

2025-10-31

Introduction

Everything in the R programming language is an object.
When we talk about data types in R, we refer to the simplest data objects we can handle — also known as R atomic data types.

Atomic Data Types

Atomic data types are the object types which you can create (atomic) vectors.

- Numeric: integer and double (real)
- Character
- Logical
- Complex
- Raw

You can check if any data object is atomic with the is.atomic() function.
This function checks for the data type of atomic vectors.

```
is.atomic(3)  # Check if numeric value 3 is atomic
```

```
## [1] TRUE
```

```
is.atomic("R CODER")  # Check if character string is atomic
```

```
## [1] TRUE
```

Check data type in R

There are several functions that can show you the data type of an R object, such as `typeof`,`mode`,`storage.mode`,`class` and `str`.

```
typeof(1)         #"double"
```

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

```
class(2)         #"numeric"
```

```
## [1] "numeric"
```

```
storage.mode(3)  #"double"
```

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

```
mode(4)          #"numeric"
```

```
## [1] "numeric"
```

```
str(5)            # num 5
```

```
##  num 5
```

However, the main use of some of them is not to just check the data type of an R object.for instance, the class of an R object can be different from the data type (which is very useful when creating S3 classes) and the `str` function is designed to show the full structure of an object. If you want to print the R data type,We recommend** using the `typeof` function.

```
x<-1
class(x)   #"numeric"
```

```
## [1] "numeric"
```

```
class(x) <- "My_class"
class(x)   #"My_class"
```

```
## [1] "My_class"
```

```
typeof(x)  #"double"
```

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

To summarize, the following table show the differences of the possible outputs when applying:`typeof`,`storage.mode`and`mode`functions.

Comparison of typeof, storage.mode, and mode in R

typeof	storage.mode	mode
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character
raw	raw	raw

There are other functions that allow you to check if the some object belongs to some data type, returning `TRUE` or `FALSE`. As a rule, these functions start with `is`. Followed by the data type. We will review all of them in the following sections.

Numeric Data Types

The numeric data type in R is composed by double (real)and integer data types. You can check if an object is numeric with the `mode` function or if the `is.numeric` function returns `TRUE`

```
mode(55)        #"numeric"
```

```
## [1] "numeric"
```

```
is.numeric(3)   #TRUE
```

```
## [1] TRUE
```

Double or real data type

The double data type in R is the representation of a double-precision numeric object. Note that, by default, all numbers are double in R and that `inf`, -inf,NaN‘,the scientific and hexadecimal notation of numbers are also doubles.

```
typeof(2)      #"double"
```

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

```
#infinite

typeof(Inf)   #"double"
```

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

```
typeof(-Inf)  #"double"
```

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

```
#Not a number

typeof(NaN)    #"double"
```

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

```
#Scientific

typeof(3.12e3)  #"double"
```

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

```
#Hexadecimal

typeof(0xbade) #"double"
```

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

If you want to check if an object is a double,you can use the `is.double` function.

```
is.double(2)# TRUE
```

```
## [1] TRUE
```

```
is.double(2.8)# TRUE
```

```
## [1] TRUE
```

Integer data type

The integer data type can be created adding an `L` to a number. This data type is useful if you want to pass some R object to a *c* or *FORTRAN* function that expects an integer value, so this data type it is not generally needed. You can check the data type with the `is.integer` function

```
y <- 2L
typeof(y)        # "integer"
```

```
## [1] "integer"
```

```
is.integer(3)    # FALSE
```

```
## [1] FALSE
```

```r
is.integer(3L)   # TRUE
```

```
## [1] TRUE
```

Logical data type

The boolean or logical data type is composed by `TRUE`, `FALSE`, and `NA` values.

```r
t <- TRUE
f <- FALSE
n <- NA


typeof(t)    # "logical"
```

```
## [1] "logical"
```

```r
typeof(f)    # "logical"
```

```
## [1] "logical"
```

```r
typeof(n)    # "logical"
```

```
## [1] "logical"
```

In addition, the function that checks whether an object is logical or not is `is.logical`.

```r
is.logical(T)      # TRUE
```

```
## [1] TRUE
```

```r
is.logical(TRUE)   # TRUE
```

```
## [1] TRUE
```

Warning: Note that you can use `T` and `F` instead of `TRUE` or `FALSE`. However, it is not recommended, because you could accidentally override `T` or `F`, but not `TRUE` or `FALSE`, as shown in the following example:

```r
T <- "Hello"
T          # "Hello"
```

```
## [1] "Hello"
```

```r
is.logical(T)  # FALSE
```

```
## [1] FALSE
```

```r
TRUE       # TRUE
```

```
## [1] TRUE
```

```r
is.logical(TRUE) # TRUE
```

```
## [1] TRUE
```

Complex data type

The complex data type is an object that includes an imaginary number (i).
As imaginary numbers are not generally used in statistics, this data type is not very common.
The function to check the data type is is.complex.

```r
1 + 3i
```

```
## [1] 1+3i
```

```
typeof(1 + 3i)       # "complex"
```

```
## [1] "complex"
```

```
is.complex(1 + 3i)   # TRUE
```

```
## [1] TRUE
```

String or character data type

Character strings are symbols, letters, words, or phrases inside double or single quotation marks. With this in mind, you can verify that some object is of type character using the `is.character` function.

```
character <- "a"

typeof(character)        # "character"
```

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

```
is.character(character)  # TRUE
```

```
## [1] TRUE
```

Note that using single or double quotation marks is equivalent.

```
typeof('R CODER')   # "character"
```

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

It is worth mentioning that the nchar function counts the number of characters inside a string, even the empty spaces.

```
nchar("A string")   # 8
```

```
## [1] 8
```

Raw data type in R

The raw data type holds raw bytes, so it is a very unusual data type. For instance, you could transform a character object or an integer numeric value to a raw object with the `charToRaw` and `intToBits` functions, respectively.

```
a <- charToRaw("R CODER")

a                  # 52 20 43 4f 44 45 52
```

```
## [1] 52 20 43 4f 44 45 52
```

```
typeof(a)          # "raw"
```

```
## [1] "raw"
```

```
b <- intToBits(3L)
typeof(b)
```

```
## [1] "raw"
```

```
is.raw(b)  #TRUE
```

```
## [1] TRUE
```

Date and Time data type in R

A calendar date in R can be represented by using function as.Date(). The Date object has class of Date, for example,

```
x <- c("14-08-1947", "06-09-1965")
y <- c("23:03:20","22:29:56","01:03:30","18:21:03")
d <- as.Date(x, "%d-%m-%Y")
t <- strptime(y, "%H:%M:%S")
format(d, "%d-%m-%Y")
```

```
## [1] "14-08-1947" "06-09-1965"
```

Note that format() is used to set how date will be displayed. Furthermore, strptime() function is used to convert a certain character representation of a date and time into another representation.

Data types coercion in R

You can coerce data types in R with the functions starting with as., summarized in the following table: | Function | Coerced data type | |—|—| | as.numeric | Numeric | | as.integer | Integer | | as.double | Double | | as.character | Character | | as.logical | Boolean | | as.raw | Raw |

Double to integer

```
a <- 3
typeof(a)  #"double"
```

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

```
a <- as.integer(a)
typeof(a)  #"integer"
```

```
## [1] "integer"
```

Logical to Numeric/Character

```
b <- TRUE
b <- as.numeric(b)
b # 1
```

```
## [1] 1
```

```
c <- FALSE
c <- as.numeric(c)
c # 0
```

```
## [1] 0
```

```
d <- TRUE
d <- as.character(d)
d # "TRUE"
```

```
## [1] "TRUE"
```

Invalid coercion example

```
as.double("R CODER")
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

Conclusion

- Atomic data types form the foundation of R programming and include numeric, character, logical, complex, and raw types.
- Type checking can be performed using multiple functions (typeof(), class(), mode(), etc.), each providing different perspectives on an object's nature.
- Numeric types are subdivided into double (default) and integer, with integers created using the L suffix.
- Type coercion using as.*() functions allows flexible conversion between types, though incompatible conversions result in NA values.

Best practices include using TRUE/FALSE instead of T/F and being cautious with automatic coercion to avoid unexpected results.