Introduction to R Data Types

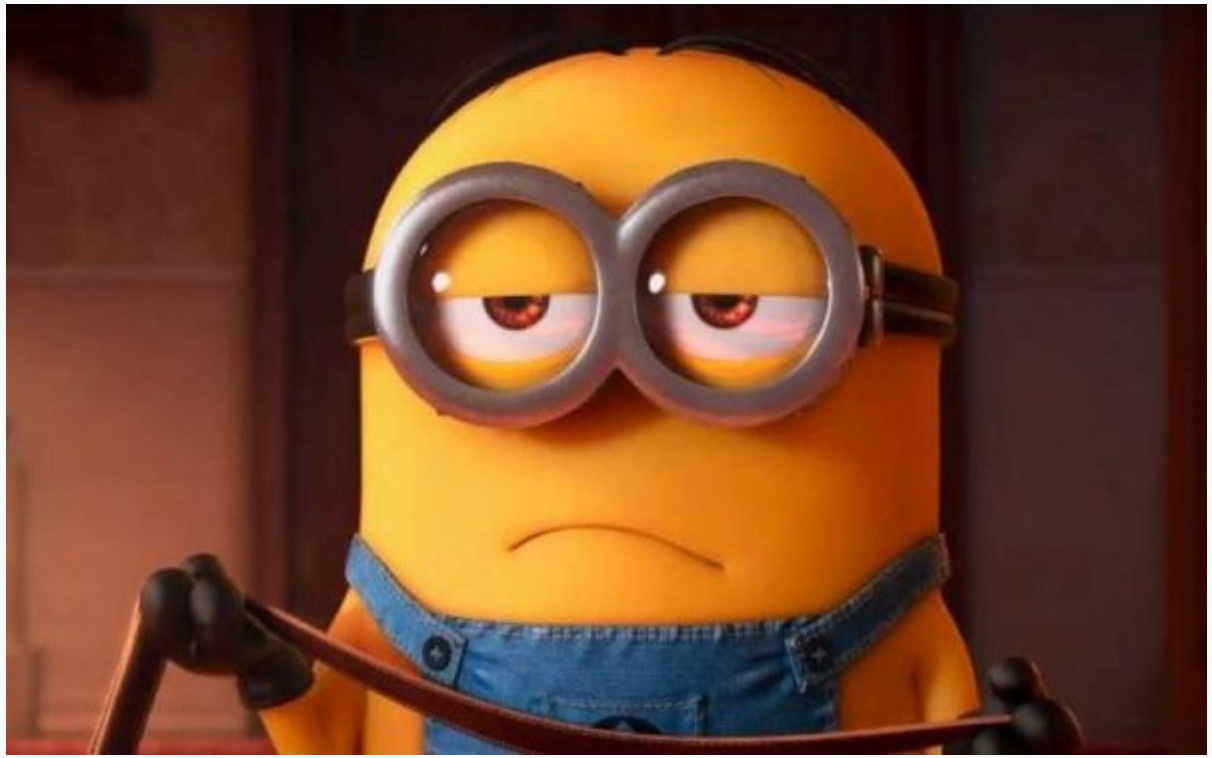
Some Boring Nuts and Bolts

Michael E DeWitt Jr

2018-09-06 (updated: 2018-09-04)

The boring bits

While the class is meant to be applied, there are some bits about the R language that are important to know when starting your programing journey.



Within R there are a limited number of data types

Data Type	Storage	Flexibility
Integer	Integer Values Only	Low
Logical	Logical Values (TRUE or FALSE)	Low
Factors	Factor Numbers	Low
Numeric	Any real number	Medium
Character	All treated as character	High

Data Types

You can inspect the data types using the `class` function:

Data Types

You can inspect the data types using the `class` function:

Integer

```
class(1L)
```

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

Data Types

You can inspect the data types using the `class` function:

Integer

```
class(1L)
```

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

Numeric

```
class(1)
```

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

Data Types

You can inspect the data types using the `class` function:

Data Types

You can inspect the data types using the `class` function:

Logical

```
class(TRUE)
```

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

Data Types

You can inspect the data types using the `class` function:

Logical

```
class(TRUE)
```

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

Character

```
class("Are we having fun yet?")
```

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

How Data Are Stored via Object Types

Within R you deal with:

Object	Dimensionality	Can Store
Atomic Vector	1	Only one type
Matrix	2	Only one type same length
Array	n	Only one type, same length
Data frame	2	One type per column, can be diff. classes, same length
List	1	Heterogeneous types, can be diff. lengths

Vectors -> The most simple case

Atomic vectors store only one "type" of data

Vectors -> The most simple case

Atomic vectors store only one "type" of data

Atomic vectors can be created using the `c` concatenate function and the assignment operator `<-`

```
my_vector <- c(1, 2, 3, 79, 22)
```

Vectors -> The most simple case

Atomic vectors store only one "type" of data

Atomic vectors can be created using the `c` concatenate function and the assignment operator `<-`

```
my_vector <- c(1, 2, 3, 79, 22)
```

Vectors can only be one type

```
class(my_vector)
```

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

Matrices -> 2 Dimensions of the Same

Matrices can store two dimensions, but the *type* and *length* must be the same

Matrices -> 2 Dimensions of the Same

Matrices can store two dimensions, but the *type* and *length* must be the same

```
(my_matrix <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

Matrices -> 2 Dimensions of the Same

Matrices can store two dimensions, but the *type* and *length* must be the same

```
(my_matrix <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
(my_matrix <- matrix(c("A","B","C","D"), nrow = 2, ncol = 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,] "A"  "B"
## [2,] "C"  "D"
```

Matrices -> 2 Dimensions of the Same

Matrices can store two dimensions, but the *type* and *length* must be the same

```
(my_matrix <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
(my_matrix <- matrix(c("A","B","C","D"), nrow = 2, ncol = 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,] "A"  "B"
## [2,] "C"  "D"
```

```
(my_matrix <- matrix(c("A",1,"C","D"), nrow = 2, ncol = 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,] "A"  "1"
## [2,] "C"  "D"
```

Arrays (n dimensions and beyond)

These are useful in simulations, but don't come up often

```
(my_3d_array <- array( 1:9, dim = c(3,3,3)))
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
```

Data Frames -> The workhorse of R

The power of a data frame is that each column can take on a different class but must be the same length

```
(my_df <- data_frame(strings = c("A", "B", "C"), numbers = 1:3,
                     logicals = c(TRUE, FALSE, TRUE)))
```

```
## # A tibble: 3 x 3
##   strings numbers logicals
##   <chr>     <int> <lgl>
## 1 A             1 TRUE
## 2 B             2 FALSE
## 3 C             3 TRUE
```

Data Frames -> The workhorse of R

The power of a data frame is that each column can take on a different class but must be the same length

```
(my_df <- data_frame(strings = c("A", "B", "C"), numbers = 1:3,
                     logicals = c(TRUE, FALSE, TRUE)))
```

```
## # A tibble: 3 x 3
##   strings numbers logicals
##   <chr>     <int> <lgl>
## 1 A             1 TRUE
## 2 B             2 FALSE
## 3 C             3 TRUE
```

We can then examine the structure of each column with the `str` function

```
str(my_df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    3 obs. of  3 variables:
##  $ strings : chr  "A" "B" "C"
##  $ numbers : int  1 2 3
##  $ logicals: logi  TRUE FALSE TRUE
```

The list -> The utility hitter

The list is the most versitile of all objects and can store anything (one-dimensionally)

```
my_list <- list(my_df = my_df, my_vector = my_vector, my_3d_array = my_3d_array, my_matrix = my_matrix)
```

The list -> The utility hitter

The list is the most versitile of all objects and can store anything (one-dimensionally)

```
my_list <- list(my_df = my_df, my_vector = my_vector, my_3d_array = my_3d_array, my_matrix = my_matrix)
```

Let's see what's inside...

```
str(my_list)
```

```
## List of 4
##  $ my_df      :Classes 'tbl_df', 'tbl' and 'data.frame':    3 obs. of  3 variables:
##   ..$ strings : chr [1:3] "A" "B" "C"
##   ..$ numbers : int [1:3] 1 2 3
##   ..$ logicals: logi [1:3] TRUE FALSE TRUE
##  $ my_vector  : num [1:5] 1 2 3 79 22
##  $ my_3d_array: int [1:3, 1:3, 1:3] 1 2 3 4 5 6 7 8 9 1 ...
##  $ my_matrix  : chr [1:2, 1:2] "A" "C" "1" "D"
```