Bite Size R

Data Types

Keith Hurley

April 12, 2019

Data Types

Anyone that has worked with data knows that every database, language, and framework seem to have their own unique set data types and R is no different. During this snack we're going to look at the main data types and data structures used by R.

Logical

In other languages, this is usually referred to as a boolean or bit or yes/no data type. In R, a piece of logical data translates to either TRUE or FALSE (all capitals - that's important!). R does allow you to shortcut by use T and F, but generally I prefer to use TRUE/FALSE to keep things more readable.

```
l<-TRUE
l
```

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

An important point in R is that logical values are allowed to be missing/null.

Integer

The simplist of the number data types in R is the integer (whole number). If you wish for R to know that a number is an integer rather than another type of number, use the "L" suffix.

```
i<-123L
i
```

```
## [1] 123
```

Numeric

The other type of number data in R is referred to as numeric. Numeric data may or may not have a decimal place. Unlike many other languages and databases, there is not differing data types for differing levels of precision (single, double, etc). All numbers in R are either integers or numeric.

```
n<-1.23
n
```

```
## [1] 1.23
```

Character Strings

Text data types in R are refered to as character data. In order for R not to intrepret your text as a variable name, all character strings must be enclosed with double quotation marks.

```
c<-"ABC 123"
c
```

```
## [1] "ABC 123"
```

Missing

Missing values in R are referred to as NA. In database terminology, this is equivelant to NULL. Missing is not actually a data type in itself, but can be assigned to data of any type. NA must be capitialized.

```
missingValue <- NA
missingValue
```

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

Not A Number

As any good math student knows, when you divide by a number by 0, you get an answer that is Not-A-Number but must be stored and used as one. In R these values are referred to by NaN (notice the lowercase a).

```
notNumber<-NaN
notNumber
```

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

Infinate

The other type of data you should be aware of is the Infinate type. This is similar to the NaN value in that it is not a missing value, but it's not truely a number either.

```
infinateNumber<-Inf
infinateNumber
```

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

WARNING

Nan and Inf values can be tricky in R. Sometimes they are considered NA values and grouped as such. Sometimes they are not. This is particularly important to understand when using the is.na and is.missing type of functions. For example, lets create a list of examples of each data type.

```
myExamples<-c(0, 1.5, 123, NA, NaN, Inf)
myExamples
```

```
## [1]   0.0   1.5 123.0    NA   NaN   Inf
```

Now let's use the is.na function to test each item in our list to see if it's a missing value.

```
is.na(myExamples)
```

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

What is returned to us is a list of logical values that tell us if each item in the list is a missing value. As you can see, both NA and NaN values are considered missing with this function while Inf values are not missing.

How about testing for NaN values?

```
is.nan(myExamples)
```

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

In this case, the function only identifies the true NaN values.

Likewise, we can test for infinate values.

```
is.infinite(myExamples)
```

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

We can use the omit function to only return values that aren't missing - but notice that it also omits NaN values.

```
na.omit(myExamples)
```

```
## [1]   0.0   1.5 123.0   Inf
## attr(,"na.action")
## [1] 4 5
## attr(,"class")
## [1] "omit"
```

Data Type Functions

For each simple data type in R, there are corresponding functions to test for type and to convert type. For example, let us look at integers.

```
is.integer(123)
```

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

Wait! What's going on here? R will assume a number is of type numeric unless you specify that it is an integer.

```
is.integer(123L)
```

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

We can test the integer we stored in a variable before.

```
is.integer(i)
```

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

If we want to convert a number to an integer, there are functions for that as well.

```
myNewInteger<-as.integer(123.45)
myNewInteger
```

```
## [1] 123
```

Let's try that again.

```
myNewInteger<-as.integer(123.78)
myNewInteger
```

```
## [1] 123
```

Notice that the as.integer function doesn't round, it simply drops the decimal part of the number. As always, we need to be mindful when converting between simple data types to maintain our data percision and integrity.

Wrap Up

That's a quick look at the different data types found in R. In a different video, we will examine how R builds on these simple data types to create complex data structures. Happy snacking!