R-datatypes-and-syntax

August 9, 2018

1 R workshop - types and syntax

- Dots in identifier names are just part of the identifier. They are not scope operators. They are not operators at all. They are just a legal character to use in the names of things.
- seq_along(x) rough equivalent of enumerate
- typeof()
- class()

1.1 Resources

- aRrgh
- Hyperpolyglot: Matlab, R , Python
- Advanced R - by Hadley Wickham
  - "According to Wickham's"tidy" approach, each variable should be a column, each observation should be a row, and each type of observational unit should be a table."
- The R Inferno - "If you are using R and you think you're in hell, this is a map for you"

2 Data types

2.1 Five Main Data Types in R

1. Atomic vector
2. Matrix
3. Array
4. List
5. Dataframe

- Everything in R is referred to as an object.
- All data in R consists of a header of metadata - the object's attributes - and the data structure itself.
- The fundamental data structure in R is the vector, which is essentially a one-dimensional array with attributes. Even the primitive data types in R are vectors. For example, 2 is a single-element vector.
- To reference a single vector element you use v[[i]] .

- To reference a sub vector you use v[i].
- For a vector v[i] and v[[i]] are almost the same thing as primitive data types are vectors.
- All arithmetic in R is vector-oriented.
- If a vector doesn't have enough elements in a vector expression, its elements are reused.
- Attributes can be used to change the way data structures are used by the system.
- The dim attribute can be used to interpret a one dimensional vector as an n dimensional array.
- A matrix is a 2 x 2 array.
- A one-dimensional vector is not the same as a one-dimensional array because it lacks a dim attribute.

3 vector: a 1-D array with homogenous datatype

- "atomic vector" - the simplest R data type.
- Linear vectors of a single primitive type
  - numeric vector - integer literals are suffixed by L
  - character vector
  - logical - TRUE, FALSE, NA means "not available"
    - aRrgh: "Do not use T and F for TRUE and FALSE. You will see people doing it but they're not your friend; T and F are just variables with default values. Set T <- F and source their code and laugh as it burns."
  - complex
- Extend the vector by assigning past the end of a vector

3.1 "Combine" Functions

- c() - combine into a vector
- cbind() combine objects as columns
- rbind() - combine objects as rows

```
In [1]: a<-c(4,5,1,3,4,5)
```

```
In [2]: class(a)
```

'numeric'

```
In [3]: a<-c(4,5,'asfd',1,3,4,5)
```

```
In [4]: class(a)
```

'character'

```
In [5]: a
```

1. '4' 2. '5' 3. 'asfd' 4. '1' 5. '3' 6. '4' 7. '5'

```
In [28]: attributes(a)
```

```
NULL
```

```
In [8]: dim( cbind( 2,3, 4,5,6) )
```

1. 1 2. 5

```
In [9]: class( cbind( 2,3, 4,5,6) )
```

'matrix'

```
In [59]: length( "hello" ) # WTF?? a character atomic vector with length 1
```

1

```
In [60]: nchar( 'hello')
```

5

4 matrix: 2-D array with optional row/column names

```
In [30]: y<-matrix(1:20, nrow=5,ncol=4)
```

```
In [31]: cells <- c(1,26,24,68)
         rnames <- c("R1", "R2")
         cnames <- c("C1", "C2")
         mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
           dimnames=list(rnames, cnames))
```

```
In [32]: mymatrix
```

	C1	C2
R1	1	26
R2	24	68

```
In [33]: attributes( mymatrix)
```

$dim 1. 2 2. 2

$dimnames 1. (a) 'R1' (b) 'R2'
2. (a) 'C1' (b) 'C2'

4.1 Reshape a matrix by assigning to dim

```
In [36]: x<-c(1,2,3,4)
         dim(x)<- c(2,2)
```

```
In [37]: x
```

```
1  3
2  4
```

5 array 3+-D array

see help(array)

```
In [44]: x<- array(0,c(2,3,4))
```

```
In [45]: x
```

1. 0 2. 0 3. 0 4. 0 5. 0 6. 0 7. 0 8. 0 9. 0 10. 0 11. 0 12. 0 13. 0 14. 0 15. 0 16. 0 17. 0 18. 0 19. 0 20. 0 21. 0 22. 0 23. 0 24. 0

```
In [49]: class( attributes(x) )
```

'list'

6 List: an ordered collection of objects

```
In [ ]: w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)
```

```
In [ ]: as.matrix(w)
```

```
In [ ]: cbind( w, 'hello')
```

```
In [50]: n = c(2, 3, 5)
         s = c("aa", "bb", "cc", "dd", "ee")
         b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
         x = list(n, s, b, 3)   # x contains copies of n, s, b
```

6.1 List Slicing with single bracket

```
In [11]: x[2]
```

1. (a) 'aa' (b) 'bb' (c) 'cc' (d) 'dd' (e) 'ee'

```
In [12]: x[2:3]
```

1. (a) 'aa' (b) 'bb' (c) 'cc' (d) 'dd' (e) 'ee'
2. (a) TRUE (b) FALSE (c) TRUE (d) FALSE (e) FALSE

6.2 List member reference using double bracket [[]]

```
In [15]: x[[2]][4] <- "ppppp"
```

```
In [16]: x
```

1. (a) 2 (b) 3 (c) 5
2. (a) 'aa' (b) 'bb' (c) 'cc' (d) 'ppppp' (e) 'ee'
3. (a) TRUE (b) FALSE (c) TRUE (d) FALSE (e) FALSE
4. 3

```
In [20]: x[2][4]
```

1. NULL

6.3 Subsetting

- [- when applied to a list, always returns a list
- [[- only returns a single value
  - use to pull pieces out of a list
  - use when the var name is stored in a variable
- $ - shorthand for [[combined with character subsetting
  - use it for partial matching

6.3.1 Integer vs Logical Subsetting

- positive integers - return elements at specified positions
- negative integers omit elements at the specified positions
- logical vectors get you the elements where TRUE

6.3.2 Simplifying vs. preserving: Comparing "[" and "[["

- Atomic vector- "[" keeps names, whereas "[[" does not:
- List - return the object inside the list, not a single element
- factor - drop any unused levels
- matrix or array - if any of the dimensions has length 1, drop that dimension
- data frame - if output is a single column, return a vector and not a data frame

```
In [1]: nx <- c(Abc = 123, pi = pi)
        nx[1] ; nx["pi"]
        nx[[1]] ; nx[["pi"]]
```

Abc: 123
pi: 3.14159265358979
123
3.14159265358979

```
In [64]: class( nx[1] )
```

'numeric'

```
In [65]: typeof( nx[1] )
```

'double'

```
In [67]: attributes( nx[1]  )
```

$names = 'Abc'

```
In [68]: class( nx[[1]] )
```

'numeric'

```
In [69]: typeof( nx[[1]] )
```

'double'

```
In [70]: attributes( nx[[1]] )
```

```
NULL
```

6.4 str() function is like glimpse

```
In [52]: str( x )
```

```
List of 4
 $ : num [1:3] 2 3 5
 $ : chr [1:5] "aa" "bb" "cc" "dd" ...
 $ : logi [1:5] TRUE FALSE TRUE FALSE FALSE
 $ : num 3
```

'NULL'

6.5 unlist() flattens list into a vector

```
In [56]: unlist( x )
```

1. '2' 2. '3' 3. '5' 4. 'aa' 5. 'bb' 6. 'cc' 7. 'dd' 8. 'ee' 9. 'TRUE' 10. 'FALSE' 11. 'TRUE' 12. 'FALSE' 13. 'FALSE' 14. '3'

6.6 Extraction operator

- The $ allows you extract elements by name from a named list
- The main difference is that $ does not allow computed indices, whereas [[does.
- see ?Extract

7 data.frame - lists of columns

8 Attributes - Object Metadata

- names, dimensions, dimnames, classes, time series attributes ## Get and set attributes with attributes() and attr()
- attributes() function returns a list
- length(): nrow() ncol() for matrices, dim for arrays()
- names(): rownames() colnames(), dimnames()

```
In [25]: attr( list, 'names')
```

```
NULL
```

9 Casting

- as.integer()
- as.character()
- as.numeric()

10 Formula data Type

- express relationship between variables
- typeof = language, class = formula
- Captures an unevaluated expression
  - The data values that have been assigned to the symbols in the formula are not accessed when the formula itself is created
  - “capture the meaning of this code without evaluating it right away.”
  - Captures the context or environment in which the expression was created. Captures the values of variables without evaluating them so they can be interpreted by the function
- Characterized by the tilde operator
  - two-sided formula
    - left hand side of tilde is dependent variable and independent variables on the right hand side
    - one-sided formula has no left side
  - check sidedness using length()
  - access elements of formula using [[]] operator for indices 1, 2, and 3 ## Symbols ### Operators built into R
-
  - * for using multiple indepent variables
-
  - * for ignoring variables
- : - for inteaction
-
  - * for crossing
- %in% - for nesting
- ^ - for limit crossing to the specified degree
- I() - the “as-is” operator - “inhibit the interpretation of operators such as”+“,”-“,”*" and “^” as formula operators, so they are used as arithmetical operators"
- . operator - everything else, all the rest of the variables in the matrix/data.frame ### Additional operators/functionality provided by 3rd party packages
- Multi-response formulas
-
- ||

10.1 Inspecting Formiulas in R

```
* terms()
* all.vars()
* update( y ~ x1 + x2, ~ . + x3 ) . # y ~ x1 + x2 + x3
```

11 Magrittr pipes

- %>%
- %$%
- . placeholder

12 foreach

- A Guide to parallelism in R

12.1 foreach + %do%

- equivalent to lapply
- nested foreach's with %:%

12.1.1 Return lists

- foreach(i=1:3) %do% sqrt(i)
- foreach(a=1:3, b=rep(10, 3)) %do% (a + b)
- Can use parens for predicate ### Return things other than lists using .combine arg
- .combine='c' makes vector
- .combine='cbind' makes matrix: Matrix foreach(i=1:4, .combine='cbind') %do% rnorm(4)

12.2 foreach + %dopar%

- tell children which packages to require using .packages arg
- Or better to be explicit and use :: scoping like dplyr::count

12.3 List comprehensions - allows you to add an if clause

- foreach(a=irnorm(1, count=10), .combine='c') %:% when(a >= 0) %do% sqrt(a)

13 doParallel

13.1 Ex. 1

```
cl <- parallel::makeForkCluster(2)
doParallel::registerDoParallel(cl)
foreach(i = 1:3, .combine = 'c') %dopar% {
  sqrt(i)
}
parallel::stopCluster(cl)
```

13.2 Ex. 2: Using doParallel::parLapply

```
library(doParallel)
no_cores <- detectCores() - 1
registerDoParallel(cores=no_cores)
cl <- makeCluster(no_cores, type="FORK")
result <- parLapply(cl, 10:10000, getPrimeNumbers)
stopCluster(cl)
```

14 Dplyr do()

- always returns a dataframe
- always needs specification of . placeholder
- use with group_by()
- can extract out of . placeholder, i.e., .$varname