R Notes

2023-05-04

> **Note**
>
> All R, math, statistics notes here.

Create a footnote: [1]

Future!

- reactR - connect js/html widgets? Need?
- Health Labs - enter Ox data
- 040 - sapply, joins, cartesian, 1-example each, cheat, see Jeremy Owens SQL table str, foreign key etc.
- tidygeocoder:: US map, simple examples?
- Review June's articles

Environemnts, Namespacds

- conflicted package.

Metaprogramming, NSE

- Hadley, Advanced R, version 1 (Expressions)
- Quoting/Eval: Lionel: https://rpubs.com/lionel-/programming-draft

[1] This is footnote one.

- https://rpubs.com/lionel-/tidyeval-introduction
- https://rpubs.com/lionel-/tidyeval-dplyr-recipes
- {{}} * https://www.tidyverse.org/blog/2019/06/rlang-0-4-0/
- https://rpubs.com/lionel-/superstache
- function masking/pkgs: https://www.r-bloggers.com/2013/09/control-the-function-scope-by-the-r-package-namescope/ (using namespace to control scop)
- {}} - https://www.r-bloggers.com/2019/06/curly-curly-the-successor-of-bang-bang-2/
- env stack - https://www.r-bloggers.com/2014/12/tips-on-non-standard-evaluation-in-r/
- quoting, dplyr - https://www.r-bloggers.com/2019/07/bang-bang-how-to-program-with-dplyr/
- 2012 namespace pkg, by Hadley, et al : https://cran.r-project.org/web/packages/namespace/index.html
- https://www.rostrum.blog/2023/03/03/getparsedata/ (parse, tokens, more advanced)
- **Examples**
- http://zevross.com/blog/2018/09/11/writing-efficient-and-streamlined-r-code-with-help-from-the-new-rlang-package/ (mostly ggplot2)
- Murdoch, R Journal vol 2/2 2010
- statisticsglob.com (several parse examples)
- advanced-r-solutions.rbind.io/expressions.html
- Tierney, codeTools ?
- **Manuals:**
- ns-load: https://stat.ethz.ch/R-manual/R-devel/library/base/html/ns-load.html
- getFromNamespace: https://stat.ethz.ch/R-manual/R-devel/library/utils/html/getFromNamespace.html
- R-exts:package-ns: https://cran.r-project.org/doc/manuals/R-exts.html#Package-namespaces
- R-ints 1.2 https://cran.r-project.org/doc/manuals/r-release/R-ints.html#Environments-and-variable-lookup R-Lang 2.1.3 Language Objects (calls, expressions, names) R-Lang 6 Compute on Language: https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Computing-on-the-language
- **stackoverflow**

- https://stackoverflow.com/questions/14988722/in-r-what-does-loaded-via-a-namespace-and-not-attached-mean
- {{}}https://stackoverflow.com/questions/62791997/what-is-the-embracing-operator
-
- **Posit**
- ns, but not in package: https://community.rstudio.com/t/is-it-dangerous-to-create-a-namespace-for-functions-in-a-script-without-a-package/91141

Useful? - wrapr https://winvector.github.io/wrapr/index.html

Will only appear in HTML.

R Definitions & Examples (TERSE)

Manuals

\section(Definitions}

a term its definition

attach - add df/list/datafile/env to search list, at a `pos` and `name` - added to R ~1988 [Add Link] - compare to `with`

explain process: capture unevaluated code; manipulate it; later evaluate

- defuse, substitution, quasiquotation,
- injection (into unevaluated expression)
- defuse & inject are opposites
- embrase {{}}, tells function must first evalute the unevaluated content inside {{}}

bindings: - link between name (symbol) and an object - EX: f (name) and the definition (function(x) {…})

Call A call is an unevaluated function, together with arguments that are

create a call

- See: R-lang 6.5, call, sys.call v match.call

```
#| label: intro_call
#| collapse: true
#| include: true
f  <- function(x) {
    x^2}
cl  <- call("f", 3)

# display f as unevaluated f(3)
cl
```

```
f(3)
```

```
##  Test for call.
    is.call(f)
```

```
[1] FALSE
```

```
    is.call("f")
```

```
[1] FALSE
```

```
    is.call(cl)
```

```
[1] TRUE
```

```
## To evaluate a call.
    eval(cl)
```

```
[1] 9
```

Like symbol, expression, call is typeof 'language'.

```
three kinds of language objects that are available for modification, calls,
expressions, and functions
r-lang 6.1
```

is.call() is T only for calls. See: "in try… 4010_match_call_examples.Rmd"

Calling environment of a function To run, a function must be called. Calling environment refers to environment of the calling function. Also called parent environment. Not to be confused with a function environment. See example ?? The `calling function` is function actually making the call. f = function()g() # f is calling function

Currying Takes a function, partially evaluates, and returns as new function. Example: f(x,y,z) evaluated at z=a, returns g(x,y) = f(x,y,a)

context internal, stack of C structs, track execution (see R Internals 1.4) Allows flow control | error reporting (traceback) | sys.* to work. (except sys.status)

Closures create context. internals do not. primitives only in special situations. (sys.frame, sys.call count closures from either end of context stack) TODO - do not understand.

Contexts are Not counted, not reported, not on stack, and coder has no access to these functions.

REF: R Internals 1.4

byte code "Readable" concise instructions; not machine code; no user access; use JIT compiler or interpeter

Deparse See **parse**.

Evaluation or execution environment In R, a function runs in an environment specific to that function. Also referred to as frame or context. This frame holds evaluation environment. The frame ends when the function completes.

Evaluation parser, evaluator

```
When a user types a command at the prompt (or when an expression is read from a
file) the first thing that happens to it is that the command is transformed by
the parser into an internal representation. The evaluator executes parsed R
expressions and returns the value of the expression. All expressions have a
value. This is the core of the language.
```

https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Evaluation-of-expressions

Expression expression() takes an R object and returns expression, which a R list. Note: a `statement` is code that may do things (side effects) but does not return anything.

Expression v Language R expressions based on list and can be broken down further. Often these are language pieces. Lanugage use pairlists.

First Class - A function can also be used as an argument to another function. Example: lapply(list(), mean)

Frame vs Environment Frame refers to the calling stack of functions. Environment, in R, is property of function, where it looks to find non-local variables.

Function, properties formals(f) arguments in function defintion body(f) code environment(f) finds values of non-formal (non-local) variables where the function was created. **Higher Order Function** Function that takes another function as an argument and …

immutable

Interactive vs. Non-Interactive R, or S, originally designed to be interactive, ie command and response at console. .R, .Rmd, Rscript, R CMD BATCH (TODO: some error conditions do not work in BATCH ??)

Lambda Calculus Instead of naming function (f(x,y) = x^2 + y^2) create abstraction. (x,y) –> x^2 + y^2. Or instead of g(x) = x + 2; write lambdax.x+2. Easy to chain

Lexical Scope How R function finds *unbound* variables: in environment where function was created. **Dynamic Scope** Method to find variables in Call Stack at runtime. R uses Lexical Scope, however the R Language allows coder to select environment to evaluate variables. (REF: June Choe Slack/Nov 3 2021)

Scheme added (?) Portion of code in which binding applies to a variable??

In R, an "evaluator" find any "unbound symbols" (in an expression) by using variable bindings in effect when created.

match - match.arg - match.call - match.fun

namespace: REF: https://cran.r-project.org/doc/manuals/r-release/R-ints.html#Namespaces | Namespaces are environments associated with packages (and once again the base package is special and will be considered separately). A package pkg defines two environments namespace:pkg and package:pkg: it is package:pkg that can be attached and form part of the search path.

operator (non-R) An operate takes a function f and returns new function g. Example: $f'(x) = g(x)$

```
# test  is this grey?
```

package:

Pairlist

Parent Frame of function If function g() is called inside body of function f, the g has the parent frame (aka calling environment) that is execution environment of f. DRAW Diagram

Parse Convert a string (character vector) into an R Expression (ie code), which is NOT a string. Motivation is to setup R object for manipulation *before* evaluation. Parse(*.R) removes comments. Note: after parsing, the result is NOT character(1), a string.

Deparse converts an R Expression to a string (character vector) .

Parse & Deparse are NOT? opposites. See Murdoch

(latex) parse: string ==> R expression (error if invalid) deparse: R expression ==> string (actually: structure(expression(), scrfile)

options Temporary vs global vs local. Read R manual.(TODO)

Primitive vs Internal function

- `if`, `+`, sin, sqrt
- C functions
- SEE ADV-R Chapter 6, code: 059 (myoldcode)
- SEE https://nsaunders.wordpress.com/2018/06/22/idle-thoughts-lead-to-r-internals-how-to-count-function-arguments/
- SEE R Internals/Ch 2
- Do not understand at deeper level

```r
#   TODO
#   R complains about putting function in data.frame

### check several functions
y  <- list(sin, "sin", c, switch, typeof, sqrt, `if`, `+`)

quote(sin)
quote("sin")
quote(c)
quote(sqrt)
quote(`if`)
quote(`+`)
deparse(y)
data.frame(object = y,
           typeof = sapply(y, typeof),
           is.primitive = sapply(y, is.primitive),
           is.function = sapply(y, is.function))
```

R R has two parents: S, based on C, Fortran for statistics. R also has functional component, based on Scheme.

```
It is possible to abuse R, using it more like S code. (?)

**Reification** Abstract idea to treat all code as "data", including functions, structures, e

::: {.content-visible when-format="html"}
##  Referencial Transparency
:::

::: {.content-visible when-format="pdf"}

\section{Referencial Transparency}
:::

A function f is \textbf{referencial  transparent} IF replacing x with its value
returns same.

::: {.cell hash='310_R_notes_cache/pdf/unnamed-chunk-4_0b3c9610d3ec45f7fc4f4602476845b2'}

```{.r .cell-code}
f = function(x) x
x = 6
```
```

```
identical(f(x), f(6))
```

```
[1] TRUE
```

:::

However, not all R functions have this property.

```
x=6
identical(quote(x), quote(6))
```

```
[1] FALSE
```

Referencial Semantics Changes to values are done in memory. There is no copy.

Substitute When used in function with formal variable, substitute stops evaluation, captures the user's code and returns a call (ie unevaluated)

syntax How code looks, is { in right place, a grammar.

Syntax Sugar Syntax to make easier for human to express or write code efficiently.

Vectorize No loops, no for, no lapply. Example:

```
a <- 1:10^4
x <- a[a %% 2 == 0]         # select elments of a vector
```

```
Example:   array[i]  vs. get_array(array, i)
Example:   `+` (1,2) vs.  1 + 2
```

semantics What does the code DO?

String String ("5+5") is NOT call. No such thing as evaluating a string. See 0210_ You can PARSE a string and then manipulate it. Simpler to eval a quote(5+5) to return the sum.

String Interpolation

Method to substitute the value of expr into a string. Can think of it as `template` with holes. SEE: https://www.r-bloggers.com/2018/03/math-notation-for-r-plot-titles-expression-and-bquote/

bquote examples SEE 410

```
x = 5
bquote(x == .(x))
```

```
x == 5
```

Symbol

Symbol

3.1.2 Symbol lookup | In this small example y is a symbol and its value is 4. A symbol is an R object too,

```
y = 4
y
```

```
[1] 4
```

```
is.symbol(y)
```

```
[1] FALSE
```

```
is.name(y)
```

```
[1] FALSE
```

```
is.object(y)
```

```
[1] FALSE
```

```
# but
y = as.symbol(y)
is.symbol(y)
```

```
[1] TRUE
```

```
y
```

`4`

See **R Lang Ref: 2.1.3.1** Symbol (aka name), usually name of R object. Use ‘as.name() to coerce to symbol or quote() or atoms of parse()

In order to manipulate symbols we need a new element in our language: the ability to quote a data object. Suppose we want to construct the list (a b). We can’t accomplish this with (list a b), because this expression constructs a list of the values of a and b rather than the symbols themselves. This issue is well known in the context of natural languages, where words and sentences may be regarded either as semantic entities or as character strings (syntactic entities). The common practice in natural languages is to use quotation marks to indicate that a word or a sentence is to be treated literally as a string of characters. For instance, the first letter of "John" is clearly "J." If we tell somebody "say your name aloud," we expect to hear that person’s name. However, if we tell somebody "say ‘your name’ aloud," we expect to hear the words "your name." Note that we are forced to nest quotation marks to describe what somebody else might say. We can follow this same practice to identify lists and symbols that are to be treated as data objects rather than as expressions to be evaluated. However, our format for quoting differs from that of natural languages in that we place a quotation mark (traditionally, the single quote symbol ’) only at the beginning of the object to be quoted. We can get away with this in Scheme syntax because we rely on blanks and parentheses to delimit objects. Thus, the meaning of the single quote character is to quote the next object. Now we can distinguish between symbols and their values:

https://stackoverflow.com/questions/8846628/what-exactly-is-a-symbol-in-lisp-scheme

Tidy Evaluation

- pronouns, to distinguish between objects in environment ls() .env*cylandnotassociatedwiththedfanddatacol*
  (df)

Variable Three kinds:
* formals,x f = function(x= …)
* local,a f = function() {a =10}
* free, unbound, global, z f = function() (print(z))

---

function

```
  f  <- function(x=NULL) {
      x^2
  }

  formals(f)       ## pairlist
```

```
$x
NULL
```

```
  body(f)          ## language, $\code{call}$
```

```
{
    x^2
}
```

```
  environment(f)  ## environment
```

```
<environment: R_GlobalEnv>
```

```
  args(f)          ## closure
```

```
function (x = NULL)
NULL
```

```
  ## returns expression
  parse(text= '2^2')
```

```
expression(2^2)
```

```
## fails, does not know a is.
# parse(text= '2a')
```

call

```
f  <- function(x=NULL) {
}

cl  <- call("f", list(x=2))
cl
```

```
f(list(x = 2))
```

```
is.function(cl)
```

```
[1] FALSE
```

```
is.call(cl)
```

```
[1] TRUE
```

```
##  Args must be evaluated, even if f is unevaluted
    x  <- 2
    call("f", list(x))
```

```
f(list(2))
```

```
    #call("f", list(x=a))   # throws error
```

```
res  <- substitute(x+a)
res
```

```
x + a
```

```
is.call(res)
```

```
[1] TRUE
```

::: #### K-nearest neighbors, K is given

$\forall x \in X$, which could be any dimension , is already assigned to a region. For a new point, examine its K nearest neighbors who decide by majority vote which region x belongs to. SEE: wine example SEE: Gaglow book.

Bias-Var Tradeoff.

With non-zero `noise`, of variance σ^2 the best approximate to $f(x)$ will always have non-zero error: Isn't there a relation betwee E, VAR? like x and p ?

$$Error = E(f_hat) + Var(f_hat) + \sigma^2$$

SEE Berkeley Crash Course; Matloff

Questions

```
#   Explain how Base R finds column name
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
data(starwars)
col = "hair_color"
sum(is.na(starwars[, eval(col) ]) )
```

```
[1] 5
```

```
  sum(is.na(starwars[, col ]) )
```

```
[1] 5
```