R Cheat Sheet: tRips and tRaps for new players

General
Trap: R error messages are not helpful
Tip: use `traceback()` to understand errors

Object coercion
Trap: R objects are often silently coerced to another class/type as/when needed.
Examples: `c(1, TRUE) # -> 1 1`
`c(1, TRUE, 'cat') # -> "1" "TRUE" "cat"`
`30 < '8'` # yields `TRUE`; 30 became "30"
Tip: inspect objects with `str(x)` `mode(x)` `class(x)` `typeof(x)` `dput(x)` or `attributes(x)`

Factors (special case of coercion)
Trap: Factors cause more bug-hunting grief than just about anything else in R (especially when string and integer vectors and data.frame cols are coerced to factors)
Tip: Learn about factors and using them.
Tip: explicitly test with `is.factor(df$col)`
Tip: use `stringsAsFactors=FALSE` argument when you create a data frame from file
Trap: maths doesn't work on numeric factors and they are tricky to convert back.
Tip: try `as.numeric(as.character(factor))`
Trap: appending rows to a data frame with factor columns is tricky. Tip: make sure the row to be appended is a presented to `rbind()` as a data.frame, and not as a vector or a list (which works sometimes))
Trap: the combine function `c()` will let you combine different factors into a vector of integer codes (probably garbage).
Tip: convert factors to strings or integers (as appropriate) before combining.

Garbage in the workspace
Trap: R saves your workspace at the end of each session and reloads the saved workspace at the start of the next session. Before you know it, you can have heaps of variables lurking in your workspace that are impacting on your calculations.
Tip: use `ls()` to check on lurking variables
Tip: clean up `with rm(list = ls(all=TRUE))`
Tip: `library()` to check on loaded packages
Tip: avoid saving workspaces, start R with the `--no-save --no-restore` arguments

The 1:0 sequence in for-loops
Trap: `for(x in 1:length(y))` fails on the zero length vector. It will loop twice: first setting x to 1, then to 0.
Tip: use `for(x in seq_len(y))`
not `for(x in 1:length(y))`
Tip: `for(x in seq_along(y))` not `for(x in y)`

Space out your code and use brackets
Trap: `x<-5` # parses as `x <- 5` not `x < -5`
Trap: `1:n-1` # -> `(1:n)-1` not `1:(n-1)`
Trap: `2^2:9` # -> `(2^2):9` not `2^(2:9)`

Vectors and vector recycling
Trap: most objects in R are vectors. R does not have scalars (just length=1 vectors). Many Fns work on entire vectors at once.
Tip: In R, for-loops are often the inefficient and inelegant solution. Take the time to learn the various "apply" family of functions. Hadley Wickham's plyr package is also worth learning and using.
Trap: Math with different length vectors will work with the shorter vector recycled
Eg: `c(1, 2, 3) + c(10, 20) # -> 11, 22, 13`
Trap: `is.vector(list(1, 2, 3)) # -> TRUE`

Vectors need the c() operator
Wrong: `mean(1, 2, 3, 4, 5, 6) # -> 1`
Correct: `mean(c(1, 2, 3, 4, 5, 6)) # -> 3.5`

Use the correct Boolean operator
Tip: `|` and `&` are vectorised - use `ifelse()` (`|` and `&` also used with indexes to subset)
Tip: `||` and `&&` are not vectorised - use `if`
Trap: `||` `&&` lazy evaluation; `|` `&` full eval
Trap: `==` (Boolean equality) `=` (assignment)

Equality testing with numbers
Trap: `==` and `!=` test for near in/equality
Eg: `as.double(8) == as.integer(8)` is `TRUE`
`isTRUE(all.equal(x, y))` tests near equality
Tip: `identical(x, y)` is more fussy

Think hard about NA, NaN and NULL
Trap: `NA` and `NaN` are valid values.
Eg: `c(1, 2) == c(1, NA) # -> TRUE, NA`
Trap: many Fns fail by default on `NA` input
Tip: many functions take: `na.rm=TRUE`
Tip: vector test for NA: `any(is.na(y))`
Trap: `x == NA` is not the same as `is.na(x)`
Trap: `x == NULL` not the same as `is.null(x)`
Trap: `is.numeric(NaN)` returns `TRUE`

Indexing ([], [[]], $)
Tip: Objects are indexed from 1 to N.
Trap: many subtle differences in indexing for vectors, lists, matrices, arrays and data.frames. Return types vary depending on object being indexed and indexation method.
Tip: take the time to learn the differences
Trap: the zero-index fails silently
Eg: `c(1, 2, 3)[c(0,1,2,0,2,3)] # -> 1,2,2,3`
Trap: negative indexes return all but those
Eg: `c(1, 2, 3, 4)[-c(1, 3)]] # -> 2, 4`
Trap: `NA` is a valid Boolean index
Eg: `c(1, 2)[c(TRUE, NA)] # -> 1, NA`
Trap: mismatched Boolean indexes work
Eg: `c(1, 2, 3)[c(T,F,T,F,T)] # -> 1, 3, NA`

Coding practice
Tip: liberally use `stopifnot()` on function entry to verify argument validity (ie. enforce programming by contract)
Tip: `<-` for assignment; `=` for list names