R Cheat Sheet: Environments, Frames and the Call Stack

Environments

1) R uses environments to store the name-object pairing between variable name and the R object assigned to that variable (assign creates pair: <-, <<-, assign())
2) They are implemented with hash tables.
3) Like functions, environments are "first class objects" in R: They can be created, passed as parameters and manipulated like any other R object.
4) Environments are hierarchically organised (each env. has a parent).
5) When a function is called, R creates a new environment and the function operates in that new environment. All local variables to the function are found in that environment (aka frame).

Code example:

```
dictionary <- function() {
    # private ... effectively hidden
    e <- new.env(parent=emptyenv())
    # use emptyenv() to stop chained lookup
    keyCheck <- function(key) # sanity chk
        stopifnot(is.character(key) &&
            length(key) == 1)
    # public ... made public by list below
    hasKey <-function(key) {
        keyCheck(key)
        exists(key, where=e,
            inherits=FALSE)
    }
    rmKey <- function(key) {
        stopifnot( !missing(key) )
        keyCheck(key)
        rm(list=key, pos=e)
    }
    putObj <- function(key, obj=key) {
        stopifnot( !missing(key) )
        keyCheck(key)
        if(is.null(obj)) return(rmObj(key))
        assign(key, obj, envir=e)
    }
    getObj <- function(key) {
        stopifnot( !missing(key) )
        keyCheck(key)
        if(!hasKey(key)) return(NULL)
        e[[key]] # also $ indexing possible
    }
    allKeys <- function()
        ls(e, all.names=TRUE)
    allObjs <- function()
        eapply(e, getObj, all.names=TRUE)
    list(hasKey=hasKey, allKeys=allKeys,
        rmKey=rmKey, getObj=getObj,
        putObj=putObj, allObjs= allObjs)
}
d <- dictionary();               # create
sapply(LETTERS, d$putObj)        # populate
d$hasKey('A'); d$allKeys()       # inspect
d$allObjs()                      # inspect
d$getObj('A')                    # retrieve
d$rmKey('A'); d$hasKey('A')      # remove
```

Code example explained

The above dictionary function returns the list at the end of the function. That list and the listed callable functions exist in the environment created when the dictionary function was called. This use of functions and lexical scoping is a poor man's OOP-class-like mechanism. The function also creates an environment (e), which it uses for its hash table properties to save and retrieve key-value pairs.

Lexical and dynamic scoping

R is a lexically scoped language. Variables are resolved in terms of the function in which they were written, then the function in which that function was written, all they way back to the top-level global/package environment where the program was written. Variables are not resolved in terms of the functions that called them when the program is running (dynamic scoping). Interrogating the function call stack allows R to simulate dynamic scoping.

Frames and environments

A frame is an environment plus a system reference to a calling frame. R creates each frame to operate within (starting with the global environment, then a new frame with each function call). All frames have associated environments, but you can create environments that are not associated with the call stack (like we did with e above).

The call stack

As a function calls a new function, a stack of calling frames is built up. This call stack can be interrogated dynamically.

```
# some call stack functions ...
sys.frame()     # the current frame
parent.frame()  # get the frame for the
                # calling function (an env)
parent.frame(1) # same as above
parent.frame(2) # get the grandparent
                # function's frame
# parent.frame(n) is the same as ...
#   sys.frame(sys.parent(n))
sys.nframe()    # the current frame number
                # (global environment = 0)
                # on the call stack
sys.call()      # returns the call (which
                # is language expression)
sys.call(-1)    # parent function's call
sys.call(1)     # the first function call
                # on the call stack down
                # from the global env.
deparse(sys.call())[[1]] # string name
                # of this function
# potential confusions ...
parent.env(sys.frame()) # lexical scoping
Sys.getenv() #Operating System environment
Sys.setenv() #as above – not an R env.
```