# R: Environments

Debashis Ghosh

October 9, 2019

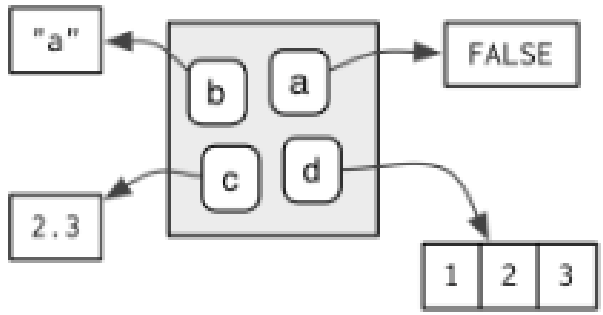
# Environments

- Environments are the data structure that enables scoping
  1. **Lexical scoping** looks up symbol values based on how functions were nested when they were created
  2. **Dynamic scoping** is used in select functions to save typing during interactive analysis (later)
- There are some resemblances between environments and lists, but also some differences.
- Environments have reference semantics; modifications happen without any sort of copying occurring.

# Environment: basics

- Environment is a collection of object names (but NOT what the objects contain).
- Example:

Figure: An environment containing the variables **a**, **b**, **c** and **d**




- Environment binds the set of object names to the values of the object.
- If the values do not have an associated name, they automatically "cleaned up" (garbage collecting)

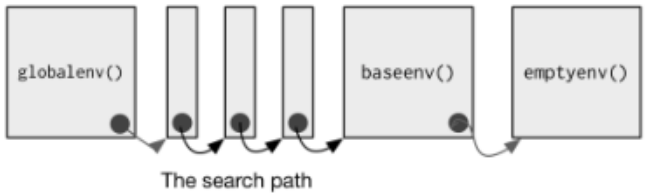
# Environments versus lists

- Similarities: can access objects using either $ or [[]]
- Differences
  1. Every object in an environment has a unique name.
  2. No ordering of objects
  3. An environment has a parent
  4. Reference semantics (different from functions)

# Parent Environments

- Environments form a recursive data structure (tree)
- Here, there is a notion of first and last
- When you call libraries, this increases the depth of the tree
- Pictorial example:

Figure: Output of **search**()




- **emptyenv()** is the one environment that does not have a parent

# Key functions with environments

- All of these functions take as an input argument an environment
- **ls.str()**: Gives output similar to what is done with **str()** and lists.
- **parent.env()**: Returns the parent environment.
- **ls()**: use the **all.names = TRUE** option to list all variables (including those that start with a period)
- **get()**: Will recursively look through environments to find an object name (in contrast to [[]] or $, which only looks in the current environment)
- **exists()**: determine if a binding exists in an environment. Can recursively search through all the higher-level environment.
- **identical()**: compare environments; returns a logical.

# Special environments

- **emptyenv()**: the ultimate ancestor of all environments; has no parent
- **globalenv()**: the interactive workspace in which you normally work
- **environment()**: current environment
- **baseenv()**: environment of the base package

# Other functions

- **search()**: lists all ancestors of the global environment
- **as.environment()**: Allows access to any ancestor environment
- **new.env()**: Creates a new environment

# Writing functions to access mutiple environments

- Because environments form a tree structure, we can use **recursion**
- Recursion accesses objects in a nested and iterative manner.

# Function environments

- When you create a function (e.g. in homework), the function creates a local environment in which your commands get executed.
- There are four types of environments associated with function creation
  1. enclosing
  2. binding
  3. execution
  4. calling

# Enclosing environment

- This is the pointer from the function to its name it is assigned and the subsequent environment that is created
- Used for lexical scoping
- Can be determined using the **environment()** function

# Binding environment

- The binding environments of a function are all the environments which have a binding to it.
- If you assign a function in a new environment, then it will be possible for enclosing and binding environments to be different.
- Package namespaces keeps the distinction between enclosing and binding environments.

# R packages

- They all have associated with them:
  1. Package environment
  2. Namespace environment
- Package environment placed on the search path
- Namespace environment contains all functions (including internal functions), and its parent environment is a special imports environment that contains bindings to all the functions that the package needs.

# Execution environments

- Related to concept of fresh start.
- If all relevant variables for the function to work are defined internally, there is no need for the function to look outside itself.
- Each time a function is called, a new environment is created to host execution.
- The parent of the execution environment is the enclosing environment of the function.
- Once the function has completed, this environment is thrown away.
- One exception: if a function is created inside of a function, then enclosing environment of the child function is the execution environment of the parent

# Calling environments

- It is possible to have variables that have different values in the environment where functions are defined relative to where they are **called**.
- Key function: **parent.frame**(). This function returns the environment where the function was called.