Cheatography

# r part5 Cheat Sheet

by Niki (worlddoit) via cheatography.com/170195/cs/35988/

## Environment 1

⛉ The environment is the data structure that powers scoping.
⛉ Scoping is the process of connecting names and values.
⛉ An environment binds a set of names to a set of values. It is essentially a disordered bag of names.
⛉ Only one environment at the time is active.
⛉ The active environment is also called the current environment.
⛉ Upon calling a function, a new environment hosts its execution and then it is garbaged (unless captured).
ϟ A **promise** contains
**1)** an expression code for the delayed computation,
**2)** an environment in which it is evaluated, and
**3)** a value accessed only by forcing

(**Basics)

## Environment 2

⛉ Functions to treat environments can be found in `rlang`
⛉ An environment is conceptually similar to a named list

```
> env1 <- rlang::env(
+ x = c(FALSE, TRUE),
+ y = "a",
+ z = 2.3,
+ t = matrix(1:4,2),
+ )
```

## Environment 2 (cont)

- Every name is unique.
- Names are not ordered.
- It has a parent.
- It is always modified in place and never copied on modify.
⛉ Track the parents up to the global

```
> rlang::env_parents(env2)
[[1]] <env: 000000000F217108>
[[2]] $ <env: global>
```

(**Basics)

## Scoping and Environments 1

**Name masking** follows from static (lexical) scoping in the environments, things defined in the current env are used:

```
> rm(list=ls())
> x<-10; y<-20; g<-function(x)
x+10
> f<-function(x) {g<-function(x)
x+1; y<-1; z<-2; r<-x+y+z+g(x)
list(current = current_env(),
caller = caller_env())}
> env <- f(x = 7)
> env_print(env$current)
<environment: 0000000006124F60>
parent: <environment: global>
bindings:
* r: <dbl>
* z: <dbl>
* y: <dbl>
* g: <fn> #this is the g defined
in the current env
* x: <dbl>
> env_print(env$caller)
<environment: global>
```

## Scoping and Environments 1 (cont)

```
parent: <environment:
package:rlang>
bindings:
* x: <dbl>
* y: <dbl>
* env: <named list>
* .Random.seed: <int>
* f: <fn>
* g: <fn>
> g
function(x) x+10
> env$current$g
function(x) x+1
<environment:
0x0000000006124f60>
> c(env$current$z,env$current$r)
[1] 2 18
```

(*** Advanced)

## Scoping and Environments 2

**Dynamic lookup names** that are not defined in the current
environment are searched in the parents

```
> rm(list=ls())
> y <- 10
> f <- function(x) {z<-2; r<-
x+y+z
+ list(current = current_env(),
caller = caller_env())
+ }
> env <- f(7)
> ls(env$current)
[1] "r" "x" "z"
> env$current$y
NULL
> ls(env$caller)
[1] "env" "f" "y"
> ls() #
[1] "env" "f" "y"
> codetools::findGlobals(f)
[1] "{" "+" "<-" "y"
```

(***Advanced)

## Scoping and Environments 3

**What happens in the function, stays in the function**

```
> rm(list=ls())
> f <- function(){a<-1;
current_env()}
> f1 <- function(){a<-2; x<<-a;
current_env()}
> f2 <- function(){a<-3; a<<-a;
current_env()}
> env <- f(); ls(env); env$a
[1] "a"
[1] 1
> ls()
[1] "env" "f" "f1" "f2"
> env1 <- f1(); ls(env1); env1$a
[1] "a"
[1] 2
> ls(); x
[1] "env" "env1" "f" "f1" "f2"
"x"
[1] 2
> env2 <- f2(); ls(env2); env2$a
[1] "a"
[1] 3
> ls(); a
[1] "a" "env" "env1" "env2" "f"
"f1" "f2" "x"
[1] 3
```

(***Advanced)

## Scoping and Environments 4

**Non-functions objects are ignored in function calls**

```
> rm(list=ls())
> x<-10; y<-20; g<-function(x)
x+10
> f <- function(x) {g<-1; y<-1;
z<-2; r<-x+y+z+g(g)
+ list(current = current_env(),
caller = caller_env())
+ }
> env<-f(7)
> ls(env$current)
[1] "g" "r" "x" "y" "z"
> ls(env$caller) #or simply ls()
[1] "env" "f" "g" "x" "y"
> env$current$g
[1] 1
> env$caller$g #or simply g
function(x) x+10
> fn_env(g)
<environment: R_GlobalEnv>
```

(***Advanced)