

7_Environments

2022-10-19

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5      v purrr  0.3.4
## v tibble  3.1.6      v dplyr  1.0.8
## v tidyr   1.2.0      v stringr 1.4.0
## v readr   2.1.2      v forcats 0.5.1
```

```
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(rlang)
```

```
##
## Attaching package: 'rlang'

## The following objects are masked from 'package:purrr':
##
##   %%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##   flatten_lgl, flatten_raw, invoke, splice
```

7 Environments

7.1 Introduction

Quiz

1. List at least three ways that an environment differs from a list.
2. What is the parent of the global environment? What is the only environment that doesn't have a parent?
3. What is the enclosing environment of a function? Why is it important?
4. How do you determine the environment from which a function was called?
5. How are <- and «- different?

7.2 Environment basics

7.2.1 Basics

```
e1 <- env(  
  a = FALSE,  
  b = "a",  
  c = 2.3,  
  d = 1:3,  
)
```

```
e1$d <- e1
```

```
e1
```

```
## <environment: 0x0000000021c97510>
```

```
env_print(e1)
```

```
## <environment: 0x0000000021c97510>  
## Parent: <environment: global>  
## Bindings:  
## * a: <lgl>  
## * b: <chr>  
## * c: <dbl>  
## * d: <env>
```

```
env_names(e1)
```

```
## [1] "a" "b" "c" "d"
```

7.2.2 Important environments

```
identical(global_env(), current_env())
```

```
## [1] TRUE
```

```
global_env() == current_env()
```

```
## Error in global_env() == current_env(): comparison (1) is possible only for atomic and list types
```

7.2.3 Parents

```
e2a <- env(d = 4, e = 5)  
e2b <- env(e2a, a = 1, b = 2, c = 3)
```

```
env_parent(e2b)
```

```
## <environment: 0x0000000020b94e30>
```

```
env_parent(e2a)
```

```
## <environment: R_GlobalEnv>
```

```
e2c <- env(empty_env(), d = 4, e = 5)
```

```
e2d <- env(e2c, a = 1, b = 2, c = 3)
```

```
env_parents(e2b)
```

```
## [[1]] <env: 0x0000000020b94e30>
```

```
## [[2]] $ <env: global>
```

```
env_parents(e2d)
```

```
## [[1]] <env: 0x0000000021b03bd0>
```

```
## [[2]] $ <env: empty>
```

```
env_parents(e2b, last = empty_env())
```

```
## [[1]] <env: 0x0000000020b94e30>
```

```
## [[2]] $ <env: global>
```

```
## [[3]] $ <env: package:rlang>
```

```
## [[4]] $ <env: package:forcats>
```

```
## [[5]] $ <env: package:stringr>
```

```
## [[6]] $ <env: package:dplyr>
```

```
## [[7]] $ <env: package:purrr>
```

```
## [[8]] $ <env: package:readr>
```

```
## [[9]] $ <env: package:tidyr>
```

```
## [[10]] $ <env: package:tibble>
```

```
## [[11]] $ <env: package:ggplot2>
```

```
## [[12]] $ <env: package:tidyverse>
```

```
## [[13]] $ <env: package:stats>
```

```
## [[14]] $ <env: package:graphics>
```

```
## [[15]] $ <env: package:grDevices>
```

```
## [[16]] $ <env: package:utils>
```

```
## [[17]] $ <env: package:datasets>
```

```
## [[18]] $ <env: package:methods>
```

```
## [[19]] $ <env: Autoloads>
```

```
## [[20]] $ <env: package:base>
```

```
## ... and 1 more environments
```

Use `parent.env()` to find the parent of an environment. No base function returns all ancestors.

7.2.4 Super assignment, `<<-`

```
x <- 0
f <- function() {
  x <- 1
}
x
```

```
## [1] 0
```

```
f()
x
```

```
## [1] 1
```

7.2.5 Getting and setting

```
e3 <- env(x = 1, y = 2)
e3$x
```

```
## [1] 1
```

```
e3$z <- 3
e3[["z"]]
```

```
## [1] 3
```

```
e3[[1]]
```

```
## Error in e3[[1]]: wrong arguments for subsetting an environment
```

```
e3[c("x", "y")]
```

```
## Error in e3[c("x", "y")]: object of type 'environment' is not subsettable
```

```
e3$xyz
```

```
## NULL
```

```
env_get(e3, "xyz")
```

```
## Error in `env_get()`:
## ! Can't find `xyz` in environment.
```

```
env_get(e3, "xyz", default = NA)
```

```
## [1] NA
```

```
env_poke(e3, "a", 100)
e3$a
```

```
## [1] 100
```

```
env_bind(e3, a = 10, b = 20)
env_names(e3)
```

```
## [1] "x" "y" "z" "a" "b"
```

```
env_has(e3, "a")
```

```
##      a
## TRUE
```

```
e3$a <- NULL
env_has(e3, "a")
```

```
##      a
## TRUE
```

```
env_unbind(e3, "a")
env_has(e3, "a")
```

```
##      a
## FALSE
```

7.2.6 Advanced bindings

```
env_bind_lazy(current_env(), b = {Sys.sleep(1); 1})
system.time(print(b))
```

```
## [1] 1
```

```
##      user  system elapsed
##      0.00    0.00    1.01
```

```
system.time(print(b))
```

```
## [1] 1
```

```
##      user  system elapsed
##         0         0         0
```

```
env_bind_active(current_env(), z1 = function(val) runif(1))
z1
```

```
## [1] 0.4010514
```

```
z1
```

```
## [1] 0.4628058
```

7.2.7 Exercises

1. List three ways in which an environment differs from a list.

There's no order to the objects, names must be unique, they have parents, are not copied when modified

2. Create an environment as illustrated by this picture.

```
e5 <- env()
e5$loop <- e5
env_print(e5)
```

```
## <environment: 0x0000000021e590f0>
## Parent: <environment: global>
## Bindings:
## * loop: <env>
```

3. Create a pair of environments as illustrated by this picture.

```
ea <- env()
eb <- env()
ea$loop <- eb
eb$dedoop <- ea
env_print(ea)
```

```
## <environment: 0x00000000215c9818>
## Parent: <environment: global>
## Bindings:
## * loop: <env>
```

```
env_print(eb)
```

```
## <environment: 0x000000002155f478>
## Parent: <environment: global>
## Bindings:
## * dedoop: <env>
```

4. Explain why `e[[1]]` and `e[c("a", "b")]` don't make sense when `e` is an environment.

for `e[[1]]` Environments have no set order so you can't index by number. `e[c("a", "b")]` would try to return two objects, but they objects can be anything which makes returning a vector of them both hard.

5. Create a version of `env_poke()` that will only bind new names, never re-bind old names. Some programming languages only do this, and are known as single assignment languages.

```
env_pokey <- function(e, name, value){
  if(!env_has(e, name)){
    env_poke(e, name, value)
  } else cat("name already exists \n")
}
```

```
aaa <- env()
aaa$a <- 21
env_print(aaa)
```

```
## <environment: 0x00000000202077c0>
## Parent: <environment: global>
## Bindings:
## * a: <dbl>
```

```
env_pokey(aaa, "a", "abcde")
```

```
## name already exists
```

```
env_print(aaa)
```

```
## <environment: 0x00000000202077c0>
## Parent: <environment: global>
## Bindings:
## * a: <dbl>
```

```
env_pokey(aaa, "b", "abcde")
env_print(aaa)
```

```
## <environment: 0x00000000202077c0>
## Parent: <environment: global>
## Bindings:
## * a: <dbl>
## * b: <chr>
```

6. What does this function do? How does it differ from `«-` and why might you prefer it?

```
rebind <- function(name, value, env = caller_env()) {
  if (identical(env, empty_env())) {
    stop("Can't find `", name, "`", call. = FALSE)
  } else if (env_has(env, name)) {
    env_poke(env, name, value)
  } else {
    rebind(name, value, env_parent(env))
  }
}
rebind("a", 10)
```

```
## Error: Can't find `a`
```

```
a <- 5
rebind("a", 10)
a
```

```
## [1] 10
```

It recursively looks for an object by name. It starts in the current environment. If it finds the object it rebinds it to the new value. If it does not, it then calls itself again using the parent of current iteration's environment. If it goes all the way to the empty environment then it prints an error saying the object can't be found.

7.3 Recursing over environments

```
where <- function(name, env = caller_env()) {
  if (identical(env, empty_env())) {
    # Base case
    stop("Can't find ", name, call. = FALSE)
  } else if (env_has(env, name)) {
    # Success case
    env
  } else {
    # Recursive case
    where(name, env_parent(env))
  }
}
```

```
where("yyy")
```

```
## Error: Can't find yyy
```

```
x <- 5
where("x")
```

```
## <environment: R_GlobalEnv>
```

```
where("mean")
```

```
## <environment: base>
```

```
e4a <- env(empty_env(), a = 1, b = 2)
e4b <- env(e4a, x = 10, a = 11)
```

```
where("a", e4b)
```

```
## <environment: 0x0000000024adb410>
```



```
where("b", e4b)
```

```
## <environment: 0x0000000024a76f00>
```

```
where("c", e4b)
```

```
## Error: Can't find c
```

```
f <- function(..., env = caller_env()) {  
  if (identical(env, empty_env())) {  
    # base case  
  } else if (success) {  
    # success case  
  } else {  
    # recursive case  
    f(..., env = env_parent(env))  
  }  
}
```

```
f2 <- function(..., env = caller_env()) {  
  while (!identical(env, empty_env())) {  
    if (success) {  
      # success case  
      return()  
    }  
    # inspect parent  
    env <- env_parent(env)  
  }  
  
  # base case  
}
```

7.3.1 Exercises

1. Modify `where()` to return all environments that contain a binding for `name`. Carefully think through what type of object the function will need to return.

```
where <- function(name, env = caller_env(), hits = list()) { # Need to carry list over each iteration  
  if (identical(env, empty_env()) & length(hits) == 0) {  
    # Base case  
    stop("Can't find ", name, call. = FALSE)  
  } else if (env_has(env, name)) {  
    # Success case  
    hits <- c(hits, env)  
    if (!identical(env_parent(env), empty_env())) {  
      where(name, env_parent(env), hits)  
    } else {  
      hits  
    }  
  } else {  
    # Recursive case  
  }
```

```

    where(name, env_parent(env), hits)
  }
}

```

```
where("a", e4b)
```

```

## [[1]]
## <environment: 0x0000000024adb410>
##
## [[2]]
## <environment: 0x0000000024a76f00>

```

```
where("a", e4a)
```

```

## [[1]]
## <environment: 0x0000000024a76f00>

```

2. Write a function called `fget()` that finds only function objects. It should have two arguments, `name` and `env`, and should obey the regular scoping rules for functions: if there's an object with a matching name that's not a function, look in the parent. For an added challenge, also add an `inherits` argument which controls whether the function recurses up the parents or only looks in one environment.

```

fget <- function(name, env, inherits = T) {
  if (identical(env, empty_env())) {
    # Base case
    stop(paste0("Can't find ", name, " in any environment"), call. = FALSE)
  } else if (env_has(env, name) && is.function(env_get(env, name))) {
    env
  } else if (inherits) {
    fget(name, env_parent(env))
  } else {
    stop(paste0("Can't find ", name, " in ", deparse(substitute(env))), call. = FALSE)
  }
}

```

```
fget(name = "mean", env = caller_env())
```

```
## <environment: namespace:base>
```

```
fget(name = "sdaafasf", env = caller_env(), inherits = T)
```

```
## Error: Can't find sdaafasf in any environment
```

```
fget(name = "fget", env = caller_env())
```

```
## <environment: R_GlobalEnv>
```

```
fget(name = "mean", env = caller_env(), inherits = F)
```

```
## Error: Can't find mean in caller_env()
```