Video 15: Scoping Quiz

Stats 102A

Miles Chen, PhD

Scoping Example 1

```
x <- 0
y <- 10
f <- function() {
  x <- 2
  y <- 100
  h <- function() { # we define h inside f
    x <- 50
    x + y
  }
  h() # we are only calling function h and this is the value it returns
}
```

Question: What will `f()` return?

Answer: What will `f()` return?

`h()` is defined in `f()`.

`h()` returns the value `x + y`.

It finds `x` in its own execution environment with value 50.

It does not find `y`, so it searches the enclosing environment which is the execution environment of funtion `f()`.

Inside `f()`'s environment, it finds `y <- 100`. It uses that and returns `50 + 100 = 150`

```
1 f()
```

```
[1] 150
```

Scoping Example 2

```
x <- 0
y <- 10
g <- function() {
  x <- 2
  y <- 100
  h() # we are only calling function h inside g
}
h <- function() { # we define h in global
  x <- 3
  x + y
}
```

Question: What will `h()` return?

Question: What will `g()` return?

Answer: What will `h()` return?

`h()` is defined in the global environment.

`h()` returns the value `x + y`.

It finds `x` in its own execution environment with value 3.

It does not find `y`, so it searches the enclosing environment which is the global environment, where `y <- 10`. It uses that and returns `3 + 10 = 13`

```
1 h()
```

```
[1] 13
```

Answer: What will `g()` return?

`g()` assigns some values in its own environment and returns the value produced by calling `h()`.

`h()` is not defined in `g()`, so R searches the higher scope and finds `h()` defined in the global environment.

`h()` returns the value `x + y`. It finds `x` in its own execution environment with value 3. It does not find `y`, so it searches the enclosing environment which is the global environment, where `y <- 10`.

It uses that and returns `3 + 10 = 13`

```
1 g()
```

```
[1] 13
```

dynamic scoping

We define a function `p()` which will use dynamic scoping.

```
x <- 0
y <- 10  # x and y are defined in the global environment

p <- function() {
  # dynamic scoping, retrieves value from calling environment
  y <- get("y", parent.frame())
  x <- 3
  x + y
}

p()
```

```
[1] 13
```

When we call `p()`, it gets the value of `y` from the calling environment, which is the global environment, (y is 10). It adds the value of x in the execution environment (x = 3) and returns 13.

dynamic scoping

Now we call `p()` from within another function `r()`. The last line of `r()` calls `p()` and returns that value.

```
r <- function() {
  x <- 2
  y <- 100
  p()
}
```

answer

```
1 r()
```

```
[1] 103
```

when we call `r()`, it defines `x` and `y` inside the execution environment of `r()`. `r()` calls `p()` inside the execution environment. `p()` gets the value of `y` from the calling environment (which is the execution environment of `r`) so it gets the value of `y = 100`. `p()` assigns 3 to value of x, and returns x + y = 103

another example

```
rm(list = ls())
a <- 1  # values in the global environment
b <- 2
l <- function() {
  print(c(a,b))
}
```

What will `l()` return?

Answer

```
a <- 1  # values in the global environment
b <- 2
l <- function() {  # this function l exists in the global env
  # when it searches for values a and b, it searches the global env
  print(c(a,b))
}
l() # prints c(1,2)
```

```
[1] 1 2
```

another example

```
a <- 1  # values in the global environment
b <- 2
m <- function() {
  a <- 4
  l()
}
```

What will `m()` return?

Answer

Recall in global: `l <- function() { print(c(a,b)) }`

```
a <- 1  # values in the global environment
b <- 2
m <- function() {
  a <- 4  # a = 4 exists in the execution environment for m
  l() # l does not exist in m. It searches and finds l in the global env
      # when l() runs, it uses the values of a and b in the global env
}
m() # prints c(1,2)
```

```
[1] 1 2
```

another example

```
a <- 1  # values in the global environment
b <- 2
n <- function() {
  a <- 4
  l <- function() {
    print(c(a,b))
  }
  l()
}
```

What will `n()` return?

Answer

Recall in global: `l <- function() { print(c(a,b)) }`

```r
a <- 1  # values in the global environment
b <- 2
n <- function() {
  a <- 4  # a = 4 exists in the execution environment for n
  l <- function() { # this l() is defined in the execution environment of n()
    print(c(a,b))
  }
  l()  # When l() runs, it uses the l() inside the environment for n()
       # It finds a in its parent env n, and uses the value of 4
       # It does not finds b in its parent env,
       # and uses the value in the global env.
}
n() # prints c(4, 2)
```

```
[1] 4 2
```

another example

```
a <- 1  # values in the global environment
b <- 2
o <- function() {
  a <- 4
  l()
  l <- function() { # we now define l() inside o()
    print(c(a,b))
  }
  l()
  b <- 5
  l()
}
```

What will be output when `o()` is run?

Answer

```r
a <- 1  # values in the global environment
b <- 2
o <- function() {
  a <- 4
  l()  # l has not yet been defined inside o().
       # So calling l() uses the l in the global env.
       # the search for a and b uses the values in the global env.

  l <- function() { # we now define l() inside o()
    print(c(a,b))
  }
  l() # now that l() has been defined, calling l() uses the l inside o()
      # and thus searching for a and b uses a in the environment of o
      # and b in the global environment.
  b <- 5
  l() # calling l() now uses the values of a and b in the environment of o
      # because they both now exist inside o.
}
o() # prints c(1, 2), then c(4, 2), then c(4, 5)
```

```
[1] 1 2
[1] 4 2
[1] 4 5
```