Lexical Scoping

Why does all this matter?

- Typically, a function is defined in the global environment, so that the values of free variables are just found in the user's workspace
- This behavior is logical for most people and is usually the "right thing" to do
- However, in R you can have functions defined *inside other functions*
  - Languages like C don't let you do this
- Now things get interesting — In this case the environment in which a function is defined is the body of another function!

Lexical Scoping

```
make.power <- function(n) {
        pow <- function(x) {
                x^n
        }
        pow
}
```

This function returns another function as its value.

```
> cube <- make.power(3)
> square <- make.power(2)
> cube(3)
[1] 27
> square(3)
[1] 9
```

Exploring a Function Closure

What's in a function's environment?

```
> ls(environment(cube))
[1] "n"   "pow"
> get("n", environment(cube))
[1] 3

> ls(environment(square))
[1] "n"   "pow"
> get("n", environment(square))
[1] 2
```

Lexical vs. Dynamic Scoping

```
y <- 10

f <- function(x) {
        y <- 2
        y^2 + g(x)
}

g <- function(x) {
        x*y
}
```

What is the value of

```
f(3)
```

Lexical vs. Dynamic Scoping

- With lexical scoping the value of `y` in the function `g` is looked up in the environment in which the function was defined, in this case the global environment, so the value of `y` is 10.
- With dynamic scoping, the value of `y` is looked up in the environment from which the function was *called* (sometimes referred to as the *calling environment*).
  - In R the calling environment is known as the *parent frame*
- So the value of `y` would be 2.

Lexical vs. Dynamic Scoping

When a function is *defined* in the global environment and is subsequently *called* from the global environment, then the defining environment and the calling environment are the same. This can sometimes give the appearance of dynamic scoping.

```
> g <- function(x) {
+ a <- 3
+ x+a+y
+ }
> g(2)
Error in g(2) : object "y" not found
> y <- 3
> g(2)
[1] 8
```

Other Languages

Other languages that support lexical scoping

- Scheme
- Perl
- Python
- Common Lisp (all languages converge to Lisp)

Consequences of Lexical Scoping

- In R, all objects must be stored in memory
- All functions must carry a pointer to their respective defining environments, which could be anywhere
- In S-PLUS, free variables are always looked up in the global workspace, so everything can be stored on the disk because the “defining environment” of all functions is the same.