Static vs dynamic scope

- Our traditional scoping rules use static, lexical scoping:
- identifiers refer to most locally-defined version, e.g. var x:
  - First see if x is defined in the current block
  - Then check the block that encloses that one
  - Then check the block that encloses that one
  - ...
  - Check to see if it is globally defined
  - Finally, give up and say x is undefined
- nesting in the (static) source code shows which x will always be used at that point in the code
- Dynamic scoping: which x is used may vary from run to run

Dynamic scope in general

- If x isn't defined inside the function it is used in, then we check if it was defined in the function that called us, and then the function that called them, etc
- Thus which variables are visible depends on who called who, which can be different from run to run:
  - Suppose f defines a local variable x=10, then calls h
  - Suppose g defines a local variable x="foo", then calls h
  - Suppose h has no local x, but prints x anyway
  - When f runs h prints 10, when g runs h prints foo

Dynamic scope in lisp

- Lisp supports dynamic scope, but we need to specify that we want a specific variable treated as dynamically scoped
- defvar variables are always dynamically scoped

```
(defvar x “global”)
(defun h ( ) (format t “using ~A~%” x))
(defun f ( ) (let ((x “f”)) (h)))
(defun g ( ) (let ((x “g”)) (h)))
(f)   ; h will print “using f”
(g)   ; h will print “using g”
(h)   ; h will print “using global”
```

Dynamic scope beyond defvar

- For local variables, if we want them to be dynamically scoped we declare them as special (for things called from that block):

```
(let  ((x 10) (y nil))
       (declare (special x))
       ... x is now dynamically scoped ...
  )
```

Example: localized dynamic scope

- p relies on a dynamically scoped y

```
(defun p ( ) (format t “using ~A~%” y))
```

- Let block with dynamically scoped y, calls p

```
(let ((y “special let”))
   (declare (special y))
   (p))  ; prints “using special let”
```

- Attempt to use p globally crashes

```
(p) ; y not special globally, p can’t find it
```