## label blocks (local functions)

- Label blocks are much like let blocks, except that we’re defining local functions instead of local variables
- We create a list of local functions which can be called from anywhere in the “body” of the label block
- These support recursive calls
- It will be very common to put a let block inside a function and then put a label block inside the let

## Small example

- getNprint uses one local function to get a value from the user, then another local function to display it

```
(defun getNprint ( )
   (let ((x nil))
      (label ; start list of local functions
         ((getvalue ( )
            (format t “Enter something: “) (setf x (read))
         (printvalue () (format t “x is ~A~%” x))) ; end of list
         (getvalue)
         (printvalue))))
```

## Recursion

- Lambda functions can’t be recursive since you can’t call them by name, but label functions can be recursive

```
(defun foo (a)
    (label ( ; start of list of local functions
        (print (n)
            (format t “~A~%” n) (if (> n 0) (print (- n 1))))
              ) ; end of list of local functions
    ; start of “body” of label block
    ; if a looks ok then call print on it
    (if (and (integerp a) (> a 0)) (print a))))
```

## let-over-lambda-over-label

- recreate our buildCircle using local functions, the lambda function can be a simple ‘dispatcher’ to call those

```
(defun circleBuilder
    (&optional (xInit 0) (yInit 0) (rInit 1))
    (let ((x 0) (y 0) (r 0))
    (label ((setCoords (cVals) ......)
            (setRad (rVal) ......)
            (getArea () .....)
            (print () .....))
```

# Body of new buildCircle

```
; after the end of the local function defs,
; initialize the local variables from the params
(if (realp xInit) (setf x xInit))
 ... etc ...
; then create the lambda “dispatch” function
(lambda (cmd &optional (arg nil))
   (cond
       ((equalp cmd ‘print) (print))
       ((equalp cmd ‘radius) (setRad arg))
       ... etc ... )))))
```

## Scoping and nesting

- The local functions aren’t visible outside the label block (just like let’s local variables aren’t visible outside the let block)
- Can nest as deeply as you like, e.g. a let inside a let inside a labels inside a let inside a labels inside a ....
- Using a clear file layout and an editor with bracket matching is a really good idea by this point!