Function definitions (defun)

- The function used to define other functions is called defun
- It expects three or more parameters:
- The first is the name of the function
- The second is the parameter list
- The remaining parameters are treated as the sequence of function calls to make (the body of the function)
- e.g. a function to return the square of x (nil for non-numbers)

```
(defun square (x) (if (numberp x) (* x x) nil))
```

Documentation strings

- It is common to make the first statement in a function simply a string, like a one-line help, e.g.

```
(defun foo (x)
     “foo just returns whatever you passed to it”
     x)
```

- To look up the documentation string for a function:

```
(documentation ‘foo ‘function)
```

Multiple statements in a function

- Not “pure” f.p., but if a function body consists of multiple statements it will execute each in sequence, then the function returns the value of the last statement run

```
(defun multByUserValue (x)
“gets a value from the user & return that * x”
   (format t “Enter a number: “)
   (* x (read)))
```

Type checking on parameters

In a function one of the first things we typically do is check the passed parameters were actually of the right types

```
(defun intpow (x y)
"returns x^y if both are integers, otherwise nil"
   (cond
      ((not (integerp x)) nil)
      ((not (integerp y)) nil
      (t (expt x y))))
```

Setf and defvar inside a function

- Variables declared with defvar are not local to the function, don’t use it inside a function (we’ll look at local vars using let blocks)
- Remember that if you use setf on an undeclared variable it acts like a defvar
- If you use setf on a parameter then it changes the local value of the parameter (generally ok, as long as that’s what you meant to do of course)

Local variables using let blocks

- Let blocks let us define and initialize a set of local variables, and use them within a sequence of lisp statements
- Let is still just a function, its return value is the value returned by the last statement in the block

```
(let
   ((a 5) (b “foo”))      ; list of local vars, a=1, b=”foo”
   (format t “b is ~A~%”) ; first statement prints “b is foo”
   (* a a))               ; last statement returns 25
```

- can be used anyplace a lisp function call can be made

Typical function layout

1st line is documentation string, rest of body is a let block with local vars, body of let is a cond, starts with error checking

```
(defun foo (a b c)
“foo does stuff”
   (let
      ((answer 42) (why “ Y!”))
      (cond
         ((equal a b) c)
         (t nil))))
```

Other options coming later

- special: for dynamically scoped variables
- &optional: to give default values to optional parameters
- &rest: to allow any number of parameters to be passed and processed
- &key: to allow keyword parameter passing instead of positional
- Values: allows a function to return multiple values (and nth-value to capture specific ones)