Symbols and properties

- Symbols are what we would generally refer to in other languages as identifiers
- In lisp we can have the code treat the symbols themselves as data, e.g. have the code look at the name of variable x, not just access the data of x, or pass the name of x to a function
- Programs can check to see if a symbol has been bound to a value or not, and can associate a list of extra properties with symbols, looking up these properties as desired

Passing symbols to functions

- If we make a function call like (f x) then lisp evaluates the value of x and passes that to f
- If we actually want to pass the symbol x, not its value, then we need to put a ‘ in front of it, e.g. (f ‘x)
- (defun f (s) ... value in s would be the symbol ‘x ...)

Using symbols as data

- Since symbols can be examined, stored, and passed to functions, we can choose to use certain symbols to have special meaning in a program, e.g.:
  - suppose we have a function that returns a list, which may be an empty list
  - if something goes wrong we want to return an error value, what should we use?
  - we can’t use nil, since that is also a valid data value
  - what about creating a symbol, e.g. ‘HorribleListError
  - The caller can check if the returned value equals ‘HorribleListError, and if not then can process normally

Binding symbols

- When we declare a function or a variable we implicitly bind the symbol to that function/variable, e.g. (defvar x 3) is binding symbol ‘x to value 3
- We can check if a symbol is currently bound to a value or a function using boundp and fboundp

```
(symbolp ‘x) ; returns t iff x is a symbol
(boundp ‘x)  ; returns t iff x is symbol bound to a value
(fboundp ‘x) ; returns t iff x is symbol bound to a func
```

Unbound

- We can ‘unbind’ a symbol from its value or function, somewhat like undeclaring a variable

```
(makunbound ‘x)
(fmakunbound ‘x)
```

Comparing symbols

- Suppose we have the names of some symbols stored in different variables, e.g. `(defvar s1 'x) (defvar s2 'y)`
- Sometimes we want to see if s1 and s2 both refer to variables with the same name
- It can be non-trivial, one reliable approach is to convert the names to strings and compare the strings, e.g.

```
(if (string= (symbol-name s1) (symbol-name s2)) ...)
```

Property lists

- We can establish a list of property names and values to associate with a symbol, rather like a hash table for the symbol (though actually has a lot more runtime overhead)

```
(get s p)  ; look up value of property p for symbol s
(setf (get s p) v) ; set a new property value, e.g.
(setf (get ‘Pi ‘hiddenType) ‘constant)
(symbol-plist s) ; get whole property list for s
(remprop s p) ; remove property p from s’s list
```