hashes

- Hashes in lisp are basically a lookup table of key-value pairs
  - can create/destroy tables
  - can add/remove key/value pairs
  - can check if key current present in table
  - can update value associated with a key
  - can look up value associated with a key
  - can iterate through the pairs based on keys
  - can collect list of values and/or list of keys
  - can run a function on every key/value pair

Basic hash table functions

- create/return an empty table, here storing in var myTable

```
(setf myTable (make-hash-table))
```

- Look up size of table (number of pairs)

```
(hash-table-count myTable)
```

- add/update key value pair, k is my key, v is new value

```
(setf (gethash k myTable) v)
```

- remove key/value pair

```
(remhash k myTable)
```

- lookup value associated with key (nil if key not present)

```
(gethash k myTable)
```

What if nil values are ok?

- `(setf result (gethash k myTable))`
- If result is nil we don’t know if value was really nil, or whether there is no key-value pair for k
- `gethash` actually returns second value, t if found, nil if not, (capture using `nth-value` or `multiple-value-bind`)

```
(multiple-value-bind (v status) (gethash k myTable)
   (if (status)
      (format t “found value ~A~%” v)
      (format t “no such key/value pair present~%”)))
```

Iterating through keys

- Special syntax set up through macros

```
(loop for k being the hash-keys of myTable do
; ... body of your loop, doing whatever with k, e.g.
(format t “next pair is ~A:~A~%” k (gethash k myTable)))
```

- Yes, that really is the actual syntax for the loop!

(assuming you want to use k as the variable for the next key and myTable is the variable containing your hash table)

Collecting lists of keys, values, or both

- Again, special syntax set up through macros
- The following returns a list of the keys of myTable

```
(loop for key being the hash-keys of myTable collect key)
```

- Or, to get a list of the values

```
(loop for key being the hash-keys of myTable collect
   (gethash key myTable))
```

- Or, to get a list of key/value pairs

```
(loop for key being the hash-keys of myTable collect
   (list key (gethash key myTable)))
```

Running a function on each pair

- The function we want to run, e.g. *myFunction*, needs to expect two parameters, the key and the value
- It will get run on every pair, but the order of pairs is not easily predictable
- We'll pass the function name and the table as parameters to *maphash* (another 'higher order' function), e.g.

```
(maphash 'myFunction myTable)
```