Step 1 (create a HashTable):

HashTables are treated for many purposes like classes or structs in C++. Let’s begin by creating a HashTable. Try the following code.

```macaulay2
myHashTable = new HashTable from 1 => "cat", B => "dog", "iii" => "mouse"
```

You can access the entries by the code:

```macaulay2
myHashTable#1
myHashTable#B
myHashTable#"iii"
```

In Macaulay2’s terminology, the terms 1, B, "iii" are called keys. The things they evaluate to are called the values. You can get the list of keys or values by the commands

```macaulay2
keys myHashTable
values myHashTable
```

HashTables can store all sorts of things. They can store lists, functions, or other HashTables.

```macaulay2
H1 = new HashTable from 2 => "string", 3 => 7
L1 = {myList => {1,2,3}, myFunc => I -> I^2, myHash => H1}
tempHash = new HashTable from L1
tempHash#myList#1
tempHash#myHash#2
R = QQ[x,y]
I = ideal(x,y)
J = (tempHash#myFunc)(I)
tempHash#myHash#2
```

Step 2 (exploring some existing objects):

Most of the objects we manipulate in Macaulay2 are (subclasses of) HashTables. You can see their internal structure by using the command peek. To see the list of valid keys (the names )

```macaulay2
peek R
peek I
peek J
phi = map(R, R, {x^2, y^2})
peek phi
M = J*R^2
peek M
```

The different objects in these HashTables are themselves HashTables. Note, you cannot actually access the objects of some of these HashTables without a little work. For example, you can do this following:

```macaulay2
phi#source
L = keys phi
otherSource = L#0
```
Explore the objects, and perhaps some other ones too. Perhaps try exploring the objects from the presentations today as well.

Some of the objects in those objects are themselves HashTables. See if you can peek inside them as well.

**Step 3 (mutable hash tables):**

Like Lists, HashTables are not mutable. That is you cannot change the entries in a HashTable. Sometimes you need to create the entries on a HashTable periodically. This is very useful if you’ve already done some computation, and want the object to remember what’s been computed in the past. Many Macaulay2 objects have a cache that stores this kind of stuff.

```plaintext
peek (J#cache)
primaryDecomposition J
peek (J#cache)
```

In fact, if you run primaryDecomposition twice on the same ideal, it won’t do the computation again, it will just read what’s already stored in the cache, which is a (CacheTable a subclass of) MutableHashTable.

You can create MutableHashTables from lists or from HashTables. And you can convert them back to non-mutable HashTables.

```plaintext
myMutableHashTable = new MutableHashTable from {1 => "cat", B => "dog", "iii" => "mouse"}
peek myMutableHashTable
myMutableHashTable2 = new MutableHashTable from myHashTable
peek myMutableHashTable2
```

You can then change the entries of a MutableHashTable like follows.

```plaintext
myMutableHashTable#1 = "hamster"
myMutableHashTable#four = "goldfish"
peek myMutableHashTable
```

Lastly, we can check if an item is in a MutableHashTable and even delete a key by the following commands.

```plaintext
myMutableHashTable#?four
myMutableHashTable#four
remove(myMutableHashTable, four)
myMutableHashTable#?four
myMutableHashTable#four
```

You might ask why you would ever use non-mutable hash tables. It turns out that they are slower for many purposes.