Recursion tutorial – extra

In this tutorial you will start to build a simple matcher. Do not look forward in the lecture series – this defeats the purpose of the tutorial.

In this work we will deliberately use a constrained equals function that only works at the atomic level, define this as...

```
(defn eq? [x y]
  (and (not (seq? x)) (not (seq? y))
       (= x y)))
```

1. write a function called *cmp* (short for compare) which compares 2 lists for equality. Use the eq? function for this – not the = function.

   (cmp '(cat mat bat) '(cat mat bat)) => {:pat (cat mat bat) :data (cat mat bat)}
   (cmp '(cat mat bat) '(cat mat bat sat)) => nil

2. enhance cmp so its <u>first argument</u> can contain a single wild card symbol (use "?" for this), so...

   (cmp '(cat ? bat) '(cat mat bat)) => {:pat (cat ? bat) :data (cat mat bat)}

3. enhance cmp further so that its <u>first arg</u> allows the symbols ?x. These should match like match variables – match like wild cards on their first use but then retain their value until the cmp function completes, so...

   (cmp '(cat ?x mat ?x) '(cat bat mat bat))
       => {:pat (cat ?x mat ?x) :data (cat bat mat bat) :?x bat}

   (cmp '(cat ?x mat ?x) '(cat bat mat sat)) => nil

4. enhance cmp even further so that it can deal with 3 matching variables ?x, ?y and ?z.