

## Solutions, Week 2, Day 1

1. Write a function `unique` that accepts a list of lists and returns a single list with only the unique elements within. That is, the argument lists may contain duplicates but the returned list does not.

```
def getUniques (l1ist):
    "l1ist is a list of lists"
    theRes = []
    for l in l1ist:
        for item in l:
            if (not (item in theRes)):
                theRes.append (item)
    return theRes
```

2. Write a function `combineDict` that takes two dictionaries and returns a single dictionary with the keys of the first and values of the second for the values of the first. That is, with two dictionaries `{'a': 'alpha'}` and `{'alpha': 1}`, the returned dictionary is `{'a': 1}`.

```
def combineDict (kdict, vdict):
    """We assume that all the arguments are fine, just
    for a simple solution."""
    theResDict = {}
    for k, v in kDict.iteritems():
        theResDict[k] = vdict[v]
    return theResDict
```

3. Describe a quick-and-dirty way of storing alignment information in a data structure.

There are lots of solutions. But a simple one could go like this: an alignment has a name or ID. It has a number of other sequences, which may have names themselves. It has a type (DNA or protein). So we could easily compose the alignment as a dictionary with keys for name, sequences, sequence names and type:

```
theAlignment = {
    'name': '?',
    'seqs': '?',
    'seqnames': '?',
    'type': '?'
}
```

The sequences and their names would obviously be lists. The alignment name and type are simply strings:

```
theAlignment = {
    'name': "CoA alignment",
    'seqs': [
        'ACCTGCATACGTACTGGCAGCAGCCCATTCATAGCGGTGCTCA',
```

```
    'ACCTGCATACGTACTGGCAGCAGCCCATTGCATAGCGGTGCTCA',  
  ],  
  'seqnames': [  
    'Vulpus vulpus',  
    'Felis domesticus',  
  ],  
  'type': "DNA",  
}
```