For this lab, we will implement a data structure called associative array using Python’s lists. Associative arrays are actually built into Python (they are called dictionaries in Python), but for this lab, you cannot use Python’s dict – you should only use lists and nested lists.

Associative arrays are similar to lists, except that instead of being able to access and store values at indices 0, 1, 2, ..., (n-1), you can store values at (basically) any index – any integer, any string, any float (though that is inadvisable), etc. Another difference is that while the length of Python lists is fixed (unless you explicitly change it), you can create new entries in an associative array without explicitly making the associative array longer (which makes sense, since the indices can be heterogeneous and are not necessarily ordered).

In Python, you are able to do something like

```
num_courses = {} # create an empty dictionary call num_courses
num_courses["EngSci"] = 6 # add an entry for the string "EngSci"
num_courses["ArtSci"] = 5 # add an entry for the string "ArtSci"
num_courses["Music"] = 5 # add an entry for the string "Music"
print(num_courses["EngSci"]) # 6
```

In associative arrays, the “indices” (e.g., "EngSci") are called keys, and the values (e.g., 6) are still called values.

For this lab, we will simulate an associative array using nested lists. For example, the data above will be stored as follows:

```
num_courses = ["EngSci", 6],
["ArtSci", 5],
["Music", 5]]
```

In other words, each entry is a list of length 2, where the first element is the key, and the second element is the value. All the entries are stored in a list. The order in which the entries are stored does not matter. To avoid ambiguity, it must never be the case that there are two entries with the same key, though it is possible for two entries to have the same value.

*Note: please don’t have functions return anything unless you are explicitly asked to do so.*

**Question 1.**

Write a function with the signature `get_value(map, key)` which takes in an associative array `map` and returns the value that corresponds to `key`. If there is no such value, the behaviour of the function is up to you.

**Question 2.**

Write a function with the signature `modify_value(map, key, value)` which takes in an associative array `map` a key `key` and a value `value`, and, depending on whether an entry with key `key` is already present in `map`, add an appropriate entry to `map` or modifies the appropriate entry in `map`.

**Question 3.**

Write a function with the signature `make_map(keys, values)` which takes in a list of keys `keys` and a list of values `values` such that for `i` in `0,1,2,..., len(keys)-1`, `[keys[i], values[i]]` is an entry in the new associative array and returns a new associative array which contains the entries specified by the lists `keys` and `values`. You can assume that the lengths of `keys` and `values` are equal.
Question 4.

Write a function with the signature `remove_entry(map, key)` which removes the entry corresponding to key `key` from the associative array `map`.

Question 5.

Write a function with the signature `get_reverse_lookup_map(map, value)` which returns a list of all the keys in the associative array `map` which correspond to value `value`.

Question 6.

Write a function with the signature `remove_by_value(map, value)` which removes all entries with the value `value` from the associative array `map`.

Question 7.

Write a function with the signature `invert_map(map)` which takes in an associative array `map` and returns an inverted version of `map`. In the new associative array, the keys are the values in `map`, and the value at key `val` is the list of the keys in `map` which correspond to `val`. For example, an inverted version of `num_courses` above would be

```
inv_num_courses = [[6, ["EngSci"]], [5, ["Music", "ArtSci"]]]
```

Question 8.

Here's a famous story about the famous mathematician Ramanujan, as told by another famous mathematician, G. H. Hardy:

I remember once going to see him when he was lying ill at Putney. I had ridden in taxi cab number 1729 and remarked that the number seemed to me rather a dull one, and that I hoped it was not an unfavorable omen. “No,” he replied, “it is a very interesting number; it is the smallest number expressible as the sum of two cubes in two different ways.”


The number 1729 is known as the Hardy-Ramanujan Number. Write a function with the signature `square_taxi_count(n)` which returns the number of numbers between 1 and `n` that are expressible as a sum of squares of positive integers in at least two different ways (note: $5 = 1^2 + 2^2 = 2^2 + 1^2$ does not count as two different ways.) You probably will want to start with writing a function which checks whether a number `k` is expressible as the sum of two squares in two different ways.