Lab 11: Logic Programming II

Learning objectives:

- Gain experience writing Prolog predicates

This labs aims to further prepare you for homework 3, which covers logic programming in Prolog.
Use the following commands to download and unpack the distribution code:

```
$ wget https://eecs390.github.io/lab/lab11/starter-files.tar.gz
$ tar xzf starter-files.tar.gz
```

1. *Sieve of Eratosthenes.* The *sieve of Eratosthenes* is a method for computing prime numbers. Given a set that initially contains all the natural numbers starting from 2, the algorithm is as follows:

   1. Let *k* be the smallest number still in the set. It must be the case that *k* is prime, so add *k* to the result.
   2. Eliminate all multiples of *k* from the set.
   3. If any numbers remain in the set, go to step 1.

   For this problem, write your code in `sieve.pl`.

   a) Write a Prolog predicate `filter_not_multiple` that relates a number and a list of numbers to another list with multiples of the number filtered out:

   ```
   ?- filter_not_multiple(2, [2, 3, 4, 5, 6, 7, 8], Filtered), !.
   Filtered = [3, 5, 7].
   ```

   b) Write a Prolog predicate `sieve` that relates a list of numbers to another list resulting from the application of the Sieve of Eratosthenes:

   ```
   ?- sieve([2, 3, 4, 5, 6, 7, 8, 9, 10, 11], Sieved), !.
   Sieved = [2, 3, 5, 7, 11].
   ```

   Your solution should use `filter_not_multiple`. Assume that the input numbers are sorted in increasing order.

   c) Write a Prolog predicate `primes` that relates an upper bound to the set of primes up to that upper bound:

   ```
   ?- primes(20, Primes), !.
   Primes = [2, 3, 5, 7, 11, 13, 17, 19].
   ```

   Use the built-in numlist predicate to generate a sorted list of numbers.

2. *List manipulation.* Complete the following Prolog list problems. You may use the built-in arithmetic and list operations. You may not use any built-in predicates. You may write helper predicates.

   a) Write a Prolog predicate `any_contains` that relates a list of lists and an item, and is true if and only if at least one of the inner lists contains the item. Examples:

   ```
   ?- any_contains([[1], [2, 3], [4]], a), !.
   false.
   ?- any_contains([[1], [2, 3], [4]], 3), !.
   true.
   ```

   Write your code in `any_contains.pl`.

   b) Write a Prolog predicate `unzip` that succeeds if its first argument is the "zip" of the second and third arguments, meaning that it interleaves the items from the second and third arguments. The second argument is required to be exactly the same length as, or one item larger than, the third argument. For example:

```
?- unzip([a, b, c, d, e, f], Evens, Odds), !.
Evens = [a, c, e],
Odds = [b, d, f].
?- unzip([a, b, c, d, e], Evens, Odds), !.
Evens = [a, c, e],
Odds = [b, d].
```

Write your code in `unzip.pl`.

Hint: You will likely find that you need two base cases, one for when the two lists have the same length and one for when the first list is one item larger.

c) Write a Prolog predicate `filter_nonempty` that relates a list of lists to another that contains only the nonempty lists from the first, preserving order. Example:

```
?- filter_nonempty([[1], [], [2, 3, 4], [], [a, b]], X), !.
X = [[1], [2, 3, 4], [a, b]].
```

Write your code in `filter_nonempty.pl`.