Homework 1: Functional Programming

---

1. Write a function **`rotate(list:int list, n:int)`** that takes a list and an integer as input, and performs a circular shift of the elements in the list in clockwise direction by n positions. For example:

```
- rotate([1,2,3,4,5],3);

[3,4,5,1,2]

- rotate([1,2,3,4,5],1);

[5,1,2,3,4].
```

Your program should handle the case where n is larger than the length of the list.

---

2. Define the function matrix_mult that takes two compatible matrices (where a matrix is represented as a list of lists) and returns the matrix that results from multiplying them. Thus:

- matrix_mult([[1,0,0],[0,1,0],[0,0,1]], [[1,2,3],[4,5,6],[7,8,9]]);

should produce the result:

[[1,2,3],[4,5,6],[7,8,9]]

---

3. Program the Tower of Hanoi puzzle. Your top level call should be of the form: **hanoi(n, peg1, peg2, peg3)** where n is the number of disks on **peg1**, and **peg1, peg2, peg3** are the parameters holding the names of the 3 pegs. **hanoi(n, peg1, peg2, peg3)** returns a list of moves where each move is a pair.

---

4. List elements do not just have to be simple values -- we can use tuples as list elements, for example. A list such as **`[(2,3.0),(5,4.4),(10,1.2)]`** is a 3-element list where each element is an **`int*real`** tuple (each list element must still be of the same type).

In this problem, we want to calculate the total mileage, total gasoline used, and our average miles per gallon for the whole trip. The trip is composed of a number of segments. So, our input is a list, each element is the mileage and gasoline used for one segment of the trip -- in an **`int*real`** tuple (the miles are just integers). You are to write a function **`mpg(trip: (int*real) list)`** that takes a list of miles*gallons tuples and returns an **`int*real*real`** tuple (a 3-tuple), of the total mileage, the total gasoline used, and the miles per gallon for the whole trip. You can (and should) define other functions to help you if you need to (hint: What are the first two values you have to return from this function? Perhaps you could write a function for each).

---

5. Suppose we represent sets (no duplicated elements) ranging over integers as lists of integers. Write ML functions for performing the following set operations:

union(S1,S2): returns the *disjoint* set-union of sets S1 and S2. The disjoint set union contains elements that are present in either S1 or S2 but not in both. E.g., disjoint union of {1,2,3} and {3,2,4,5} is {1,4,5}.

intersection(S1,S2): returns set-intersection of sets S1 and S2.

setdiff(S1,S2): returns the set-difference of sets S1 and S2

subset(S1,S2): returns true if S1 is a subset of set S2, false otherwise.

powerset(S): returns the powerset of set S as a list of lists.

---

6. Program quicksort in ML; For the pivot, you should use the element in the list that is closest to the average of all the elements in the list to be sorted. Top level call should be **qsort(l)** which will return the list **l** sorted in ascending order.

---

7. Program bubblesort in ML; Your code should be efficient, in particular, once the largest element has been "bubbled" to the end of the list, it should not be involved in any comparisons. Top level call should be **bubblesort(l)** that returns **l** sorted in ascending order.

---

8. Program the tail recursive versions of the reverse and sum functions discussed in class. Use the accumulator trick to write an O(n) function for computing Fibonacci numbers. The top levels calls are as follows:

reverse(l) returns a list that is the reverse of **l**

sum(l) returns the sum of numbers in **l**

fib(n) returns the **n**th Fibonacci number
