Folding

Folding is a technique for aggregating values drawn from complex data structures in ML (where "complex" means "made from multiple pieces") like lists and trees. Folding can be used to obtain a value for any operation that can be performed by structural recursion on a data structure.

Problem statement: A `date` is a value of type `int*int*int`, where the 1st number is a year, the 2nd number is a month, and the 3rd number is a day. Write a function `number_in_month` that takes a list of dates and a month (an `int`) and returns how many dates are in the list for the given month. Use `fold` to solve this problem.

Since `fold` is likely new to you, let's start by thinking about what the task looks like without `fold`. Here is a recursive solution.

```
let rec number_in_month dates month =
  match dates with
  | [] -> 0
  | (_,m,_)::dates' ->
    (if m = month then 1 else 0) + (number_in_month dates' month)
```

To see how this fits into the `fold` pattern, note the presence of: (1) a base case, and (2) an accumulated value.

In the base case, we have an empty list, so clearly, there are no matching months and we return 0.

In the inductive case, we either have a matching month or we do not. If the month matches, add one, otherwise, add 0. In the above code, we do not explicitly maintain an accumulator variable, but we could have. Instead the sum is maintained implicitly by building a stack of additions using recursion.

Note the use of the pattern `((_,m,_)::dates)` to extract the month `m` from the list of dates. The use of `_` means that, structurally, an element must appear (i.e., there must be a three-tuple `(_,m,_)`), but that the only value we want to bind to a variable is the middle element, `m`.

Recall the definition for `List.fold`:

```
f: ('a -> 'b -> 'a) -> initial_value: 'a -> xs: 'b list -> 'a
```

In plain language, `List.fold` is a function that takes a function `f`, an initial value, and a list of values. The function `f` should be a function that takes an accumulator value (of the same type as the initial value) and an element of the list.

Let's refactor our `number_in_month` function to use `List.fold`.

```
let number_in_month dates month : int =
  List.fold (fun acc (_,m,_) ->
    acc + (if m = month then 1 else 0)
  ) 0 dates
```