Isabelle/HOL Exercises
Lists

Summation, Flattening

Define a function `sum`, which computes the sum of elements of a list of natural numbers.

```
primrec sum :: "nat list ⇒ nat" where
  "sum []     = 0"
| "sum (x#xs) = x + sum xs"
```

Then, define a function `flatten` which flattens a list of lists by appending the member lists.

```
primrec flatten :: "'a list list ⇒ 'a list" where
  "flatten []       = []"
| "flatten (xs#xss) = xs @ flatten xss"
```

Test your functions by applying them to the following example lists:

```
lemma "sum [2::nat, 4, 8] = x"
  apply simp  — x = 14
⋮

lemma "flatten [[2::nat, 3], [4, 5], [7, 9]] = x"
  apply simp  — x = [2, 3, 4, 5, 7, 9]
⋮
```

Prove the following statements, or give a counterexample:

```
lemma "length (flatten xs) = sum (map length xs)"
  apply (induct "xs")
  apply auto
done

lemma sum_append: "sum (xs @ ys) = sum xs + sum ys"
  apply (induct "ys")
    apply simp
  apply (induct "xs")
  apply auto
done
```

```
lemma flatten_append: "flatten (xs @ ys) = flatten xs @ flatten ys"
  apply (induct "ys")
    apply simp
  apply (induct "xs")
  apply auto
done

lemma "flatten (map rev (rev xs)) = rev (flatten xs)"
  apply (induct "xs")
  apply (auto simp add: flatten_append)
done

lemma "flatten (rev (map rev xs)) = rev (flatten xs)"
  apply (induct "xs")
  apply (auto simp add: flatten_append)
done

lemma "list_all (list_all P) xs = list_all P (flatten xs)"
  apply (induct "xs")
  apply auto
done

lemma "flatten (rev xs) = flatten xs"
  quickcheck
⋮
```

A possible counterexample is: xs = [[0], [1]]

```
lemma "sum (rev xs) = sum xs"
  apply (induct "xs")
  apply (auto simp add: sum_append)
done
```

Find a (non-trivial) predicate *P* which satisfies

```
lemma "list_all P xs ⟶ length xs ≤ sum xs"
lemma "list_all (λx. 1 ≤ x) xs ⟶ length xs ≤ sum xs"
  apply (induct "xs")
  apply auto
done
```

Define, by means of primitive recursion, a function *list_exists* which checks whether an element satisfying a given property is contained in the list:

```
primrec list_exists :: "('a ⇒ bool) ⇒ ('a list ⇒ bool)" where
```

```
  "list_exists P []      = False"
| "list_exists P (x#xs) = (P x ∨ list_exists P xs)"
```

Test your function on the following examples:

```
lemma "list_exists (λ n. n < 3) [4::nat, 3, 7] = b"
  apply simp   — b is false
⋮

lemma "list_exists (λ n. n < 4) [4::nat, 3, 7] = b"
  apply simp   — b is true
⋮
```

Prove the following statements:

```
lemma list_exists_append:
  "list_exists P (xs @ ys) = (list_exists P xs ∨ list_exists P ys)"
  apply (induct "ys")
    apply simp
  apply (induct "xs")
  apply auto
done

lemma "list_exists (list_exists P) xs = list_exists P (flatten xs)"
  apply (induct "xs")
  apply (auto simp add: list_exists_append)
done
```

You could have defined `list_exists` only with the aid of `list_all`. Do this now, i.e. define a function `list_exists2` and show that it is equivalent to `list_exists`.

```
definition list_exists2 :: "('a ⇒ bool) ⇒ ('a list ⇒ bool)" where
  "list_exists2 P xs == ¬ list_all (λx. ¬ P x) xs"

lemma "list_exists2 P xs = list_exists P xs"
  apply (induct "xs")
  apply (auto simp add: list_exists2_def)
done
```