

# Exercise 7.2

Give a pattern-matching definition of a function which:

1. adds together the first two integers in a list, if a list contains at least two elements;
  2. returns the head element if the list contains one
  3. returns zero otherwise.
-

# Exercise 7.6

Define the functions 'and2' and 'or2' on a list of Booleans

These should return the conjunction or disjunction on a list of booleans

Note that these are already defined in the prelude, so call them and2 and or2

**Example:**

```
and2 [True, False] = False
```

```
or2 [False, True] = TRUE
```

-----

# Exercise 7.8

Using primitive recursion over lists, define a function:

```
elemNum :: Integer -> [Integer] -> Integer
```

so that `elemNum x xs` returns the number of times that `x` occurs in the list `xs`.

Can you define `elemNum` without primitive recursion, using list comprehensions and built-in functions instead?

# Exercise 7.9

- Define a function :

`unique :: [Integer] -> [Integer]`

so that `unique xs` returns the list of elements of `xs` which occurs exactly once.

Example:

`unique [4, 2, 1, 3, 2, 3] = [4, 1]`

- You might like to think of two solutions to this problem: one using list comprehensions and the other not.

# iSort

To insert an element at the right place into a sorted list.

```
iSort :: [Integer] -> [Integer]
iSort []          = []
iSort (x:xs)      = ins x (iSort xs)
```

```
ins :: Integer -> [Integer] -> [Integer]
ins x []        = [x]
ins x (y:ys)
  | x <= y       = x:(y:ys)
  | otherwise    = y : ins x ys
```

# Exercise 7.16

- By modifying the definition of the `ins` function we can change the behaviour of the `sort`, `iSort`. Redefine `ins` in two different ways so that
  - 1 : the list is sorted in descending order;
  - 2: duplicates are removed from the list.

-----

# Exercise 7.25

- One list is a sublist of another if the elements of the first occur in the second, in the same order. For instance, "ship" is a sublist of "Fish & Chips", but not of "hippies".

A list is a subsequence of another if it occurs as a sequence of elements next to each other.

- For example, "Chip" is a subsequence of "Fish & Chips", but not of "Chin up".
- Define functions which decide whether one string is a sublist or a subsequence of another string.

# Exercise 7.33

Define a function `isPalindrome` which tests whether a string is a palindrome.

Example of a palindrome: “Madam I’m Adam”

Note that punctuation and white space are ignored, and that there is no distinction between capital and small letters.

-----



## Exercise 7.34

Design a function:

```
subst :: String -> String -> String -> String
```

so that:

```
subst oldSub newSub st
```

is the result of replacing the first occurrence in `st` of the substring `oldSub` by the substring `newSub`. For instance:

```
subst "much " "tall " "How much is that?" = "How tall is that?"
```

-----