Q1. [20 points]

In statistics it is common to compute the variance of a sequence of real numbers\(^1\). Given a sequence of (real) numbers \(x_1, x_2, \ldots, x_n\) the mean is defined to be the average value, which is given by the formula

\[
\mu = \frac{\sum_{i=1}^{n} x_i}{n}.
\]

The variance is defined by the formula

\[
\nu = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}
\]

in words, the average of the squares of the differences between the given values and the mean value. Write an F# program to compute the variance of a given list of real numbers. You may assume that the list is not empty and you need not write error handling code: you need to learn more about F# to do it nicely. On the web page we have written a little of the code to start you off.

You should have auxiliary functions called: (i) \texttt{sumlist} which computes the sum of a list of floats, (ii) \texttt{squarelist} which produces a list of the squares of every element of a given list of floats, (iii) \texttt{mean} which computes the mean of a list of floats, (iv) \texttt{mean_diffs} that computes the list of

\(^1\)This is a very special case of computing the variance of a random variable; you do not have to know the general concept for this question.
differences of every value from the mean and finally (v) a function `variance` that computes the variance. The types are shown below:

```fsharp
val sumlist : l:float list -> float
val squarelist : l:float list -> float list
val mean : l:float list -> float
val mean_diffs : l:float list -> float list
val variance : l:float list -> float
```

Q2. [20 points]
Implement in F# a function that tests whether an element is a member of a given list. Here is the type I want.

```fsharp
val memberof : 'a * 'a list -> bool when 'a : equality
```

Here are examples of the function in action.

```fsharp
> memberof 1 [1;2;3];;
error FS0003: This value is not a function and cannot be applied

> memberof (1,[1;2;3]);;
val it : bool = true

> memberof (1, []);;
val it : bool = false

> memberof (1, [2;3;4]);;
val it : bool = false
```

Implement a function `remove` that takes an element and a list and removes all copies of the element from the list. If the element is not in the list the function should return the same list. The type should be as follows:

```fsharp
val remove : 'a * 'a list -> 'a list when 'a : equality
```

```fsharp
> remove (2, [1;3;2;4]);;
val it : int list = [1; 3; 4]
```

Q3. [15 points] Write a function `isolate` that takes a list and makes a list with only one copy of each element of the original list. I do not care if the order is scrambled in the process.

```fsharp
val isolate : l:'a list -> 'a list when 'a : equality
```

```fsharp
> isolate [4;2;4;1;6;3;5;4;1;4;2];;
val it : int list = [4; 2; 1; 6; 3; 5]

> isolate ["the";"boy";"greeted";"the";"big";"boy"];;
val it : string list = ["the"; "boy"; "greeted"; "big"]
```

Q4. [20 points]
Write a function `common` that takes a pair of lists and forms a new list containing a unique copy of each element that occurs in both lists. Here is the type and an example.

```fsharp
val common : 'a list * 'a list -> 'a list when 'a : equality
```

```fsharp
> isolate [4;2;4;1;6;3;5;4;1;4;2];;
val it : int list = [4; 2; 1; 6; 3; 5]

> isolate ["the";"boy";"greeted";"the";"big";"boy"];;
val it : string list = ["the"; "boy"; "greeted"; "big"]
```
The mergesort algorithm is a recursive algorithm for sorting lists which runs in time $O(n \log n)$. The items in the list must have an order relation defined on them, otherwise sorting does not make sense of course.

The idea is as follows: the given list $l$ is split into two equal (if the length of $l$ is odd then one of the “halves” is one item longer than the other) lists $l_1$ and $l_2$. These lists are sorted recursively and then the results are merged back to give a single sorted list. Code this in F#. Your algorithm can use $<$ as a comparison operator. Your code must have a function split that produces a pair of lists, a function merge that merges sorted lists and a function mergesort that implements the overall algorithm.

```fsharp
def split : l:'a list -> 'a list * 'a list
def merge : 'a list * 'a list -> 'a list when 'a : comparison
def mergesort : l:'a list -> 'a list when 'a : comparison
```

Example:

```fsharp
> mergesort [15 .. -2 .. 1];;
val it : int list = [1; 3; 5; 7; 9; 11; 13; 15]
> split [15 .. -2 .. 1];;
val it : int list * int list = ([15; 11; 7; 3], [13; 9; 5; 1])
```