Task 1:

Part 1:

Let's start by looking at what foldback takes as input. The first type of input we meet is: $('a \rightarrow' b \rightarrow' b)$

We know this input from the code presented that it must be the function part of foldback (also called the accumulator function) that takes this type. So, let's investigate why the accumulator function you give foldback must have this type. The definition of foldback is that it applies a function to an initial condition and the last element in the list given. Now it continues by applying the function with the remainder of the step just mentioned and the next element in the list, and so on. Thereby the function always takes two inputs. In the first step, it takes the start value as input and the last element in the list provided. The input can be general, int, floats, etc. works fine since a function can do many kinds of operations between e.g. a start value and an element from a list.

Now this description fits with the type met: $('a \rightarrow' b \rightarrow \cdots)$. We see that we can have two inputs of the general type.

The input of the function used in foldback has now been covered. But what about the output of the accumulator function? The declaration of foldback tells us that the accumulator function must take the type of 'b (so the same general type as one of the inputs). The output is the same type as the input where the start value "e" is and the remainder[1] from every time the function is applied. This, however, there is a reason for. It makes sure that the remainder type (output of the accumulator function) does not change while foldback is being used. Thereby 'a can be applied correctly repeatedly. In some cases, the foldback function would crash because of type error, if the remainder/start value had been a different type than the output. Thereby the output is 'b of the accumulator function one uses in foldback.

Continuing, we now look at the next type. This is 'a list. It makes good sense for the type to be 'a list, since it is a general list we input besides the accumulator function discussed earlier. However, it is important to note that "'a" fits with the accumulator functions type. Recall that the function had the type $('a \rightarrow' b \rightarrow' b)$ and as we discussed the elements from the list that is being input is of type 'a. Therefore the type " 'a list " is the most correct and general type. So, the 'a from 'a list matches the accumulators 'a input.

Moving on, we have now discussed the following part of types: $('a \rightarrow' b \rightarrow' b) \rightarrow 'a\ list$. The next type is 'b. This type represents the "e" (start value) that foldback takes as input. Not much needs to be stated here, as we have already discussed why the start value must have the 'b type. (It needs to be of the same type as the remainder that is used repeatedly in foldback, for the accumulator function to work properly.)

Finally, we see the output to be of type 'b: $('a \rightarrow' b \rightarrow' b) \rightarrow' a\ list \rightarrow 'b \rightarrow 'b$. Why is this the most general type? Looking back at the accumulator function (the one foldback takes as input), when foldback executes its last step to reach a final output, the function outputs a value of 'b type. Thereby the output of foldback is bounded to this and it makes sense that the most general type for output is 'b. Thereby the most general type has been declared.

[1] When we are referring to "remainder" of foldback, we are referring to the output every time the accumulator function is applied to a given list.

Part 2:

Fold:

We are dealing with the expression: fold (fun a x -> x-2*a) 0 [1;2;3]. We want to make an evaluation of this expression.

Fold has the formula: $f(\dots(f(f\ e\ x_0)x_1)\dots)x_{n-1}$, this will come in handy when we want to express the evaluation. The formula states that we must apply the function "f", to initial conditions "e" and first element of a list "x_0". After the first step, the function "f" is now applied to the product of the first step (which we called remainder in the first part), and the second element in the list, and so on. Keeping this in mind we can make an evaluation of the expression stated:

1. List.fold (fun a x -> x-2*a) 0 [1;2;3] {What we can learn here is that the function we are dealing with is f(x)=x-2*a, it starts at the initial condition of e=0, and the list of elements that is being used ($[x_0, x_1, x_2]$=[1;2;3]). "a" is the same as "e0" and is therefore the initial condition (in the first step). After the first step, "a" is the remainder from every previous step. Later, f(x) is referred to, so keep it mind that f(x)=x-2*a}

2. Step 1: fold f $e_0[x_0, x_1, x_2]$ => fold f(x) 0 [1;2;3] => $1 - 2 * 0 = 1$ | $e_0 = 0$
3. Step 2: ⇝ fold f e_1 $[x_1, x_2]$ => fold f(x) 1 [2;3] => $2 - 2 * 1 = 0$ | $e_1 = f(x)e_0\ 1 = 1 - 2 * 0 = 1$
4. Step 3: ⇝ fold f e_2 $[x_2]$ => fold f(x) 0 [3] => $3 - 2 * 0 = 3$ | $e_2 = f(x)\ e_1\ 2 = 2 - 2 * 1 = 0$
5. Step 4: ⇝ fold f $e_3[\ ]$ = e_3 => fold f(x) 3 [ ] = 3 | $e_3 = f(x)e_2\ 3 = 3 - 2 * 0 = 3$

The final answer is therefore 3. The way the evaluation is made, is by showing the general structure of fold in every step, and then translating these general steps to our case. The e's shown on the right side, is as mentioned our "a" in the function that fold takes.

Foldback:

This time we are dealing with the expression: foldback (fun x a -> x-2*a) 0 [1;2;3]. We have the same initial condition and list as before. However, foldback is a different formula: $g(x_0(x_1(\dots(g\ x_{n-1}e)\dots))$. So foldback takes the last element in the list presented and starts applying the function to that element. However, since foldback builds a large expression first, we must do this to start with. Afterwards we simplify the large expression:

1. We use the same assumptions as before about f(x), "a" and the list.

2. Step 1: foldback f(x) [1;2;3] 0
3. Step 2: ⇝ $f(x)\ 1\ (foldback\ f(x)\ [2;3]\ 0)$
4. Step 3: ⇝ $f(x)\ 1\ (f(x)2\ (foldback\ f(x)[3]0))$
5. Step 4: ⇝ $f(x)\ 1\ (f(x)\ 2\ (f(x)\ 3\ (foldback\ f(x)\ [\ ]\ 0)))$
6. Step 5: ⇝ $f(x)\ 1\ (f(x)\ 2(\ f(x)\ 3\ 0\ )))$
   Now we evaluate the expression above, starting from the inside.
7. Step 6: ⇝ $f(x)\ 1\ (f(x)\ 2(\ f(x)\ 3\ 0\ ))) = f(x)\ 1\ (f(x)\ 2(\ 3 - 0\ )))$ | $f(x)\ 3\ 0 = 3 - 2 * 0 = 3$
8. Step 7: ⇝ $f(x)\ 1\ (f(x)\ 2\ 3)) = f(x)\ 1\ (-4))$ | $f(x)\ 2\ 3 = 2 - 2 * 3 = -4$
9. Step 8: ⇝ $f(x)1\ (-4) = 9$ | $f(x)\ 1\ \ -4 = 1 - (2 * -4) = 9$

And so, after 8 steps, where we applied the foldback function and evaluated the expression, we see that the answer is 9.