In many programming languages, to declare a function you need to give it a name. Not so in functional languages. In scheme, the term 

\( (\text{lambda } (x) \ x) \)

denotes the identity function. In Haskell, the identity function is written as:

\( (\ \! x \to x) \)

In ML, the \texttt{fun} construct works like \texttt{lambda} does in scheme.

\( \text{fun } x \to \text{<body>} \)

is the form of the ML expression whose value is a function so,

\( \text{(fun } x \to x) \)

is the identity function in expressed in ML.

Now, consider a point-wise update function which, given two arguments, a function \( f \) and a pair \((x,v)\), returns a function that behaves as \( f \) does except that on input \( x \) it returns the value \( v \).

\( \text{let update } (x,v) f = (\text{fun } y \to \text{if } x = y \text{ then } v \text{ else } f y);; \)

1.) Write a function \texttt{pupdate} that returns a function but, instead of updating a single point, takes a predicate (a function from the domain of \( f \) to \texttt{bool}) that specifies the points to be updated, along with a function (say \( g \)) that is used to compute a new value if the input satisfies the predicate.

There are some examples of the expected behavior linked on the webpage under hw3\_expected.txt.

Like \texttt{update}, the new function should – return a function that, on input \( x \) behaves like \( f \) if \( p \ x \) is false and behaves like \( g \) if \( p \ x \) is true. So, a mathematical definition would be as follows:

\[
pupdate \ f \ p \ g \ x = \begin{cases} 
  g(x) & \text{if } p(x) \\
  f(x) & \text{otherwise}
\end{cases}
\]

Your new function should have the following type.

\( \text{val pupdate : ('a -> 'b) -> ('a -> bool) -> ('a -> 'b) -> 'a -> 'b = <fun>} \)

2.) Define \texttt{update'} which has the same type as \texttt{update} and has the same behavior - but define it in tems of \texttt{pupdate} i.e. make a definition that looks something like the following:

\( \text{let update'} f (x,v) = \text{pupdate ...} \)