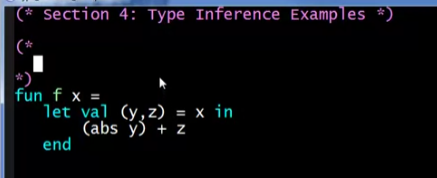
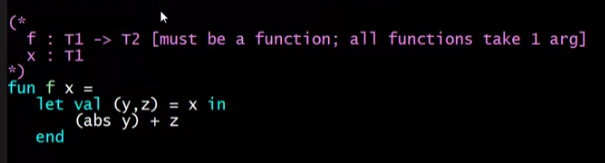


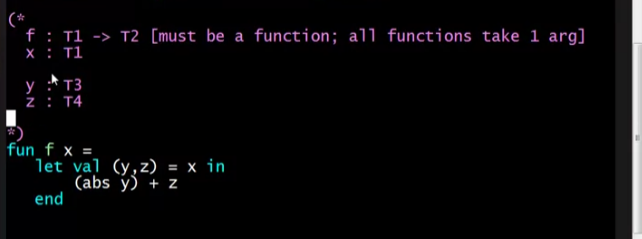
First Example:



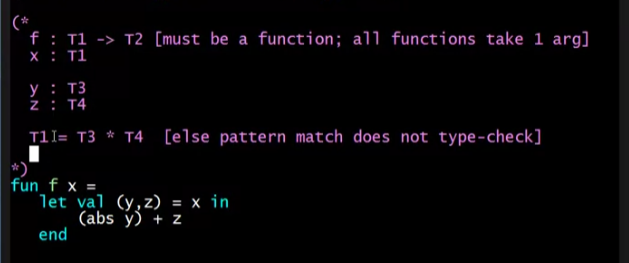
Function definition:



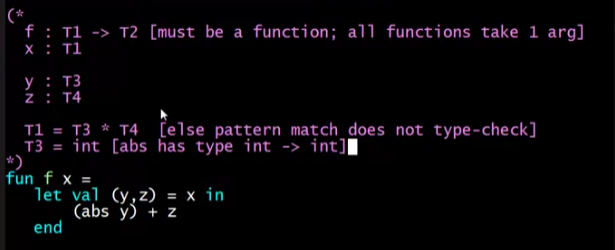
Function body:



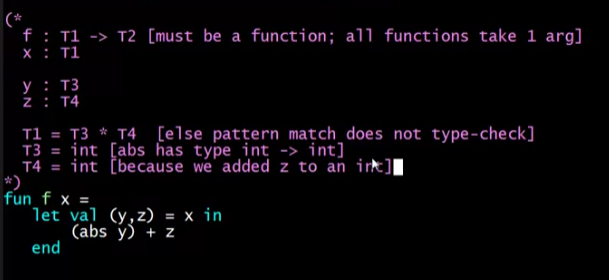
T1 type



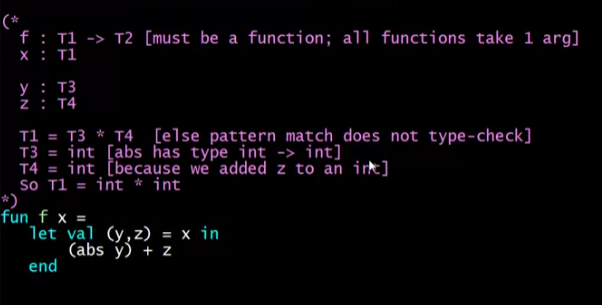
(abs y) for y : T3. T3 must be int



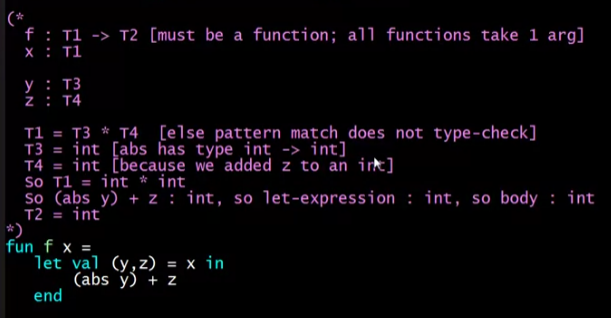
z : T4. T4 must be int



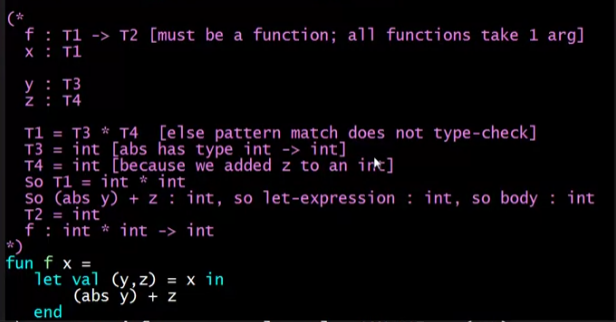
So T1: (argument type)



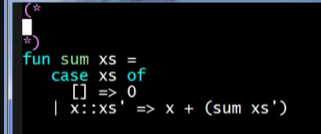
Then T2: (result/return type)



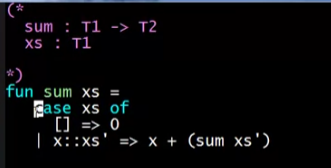
Then final type of f is:



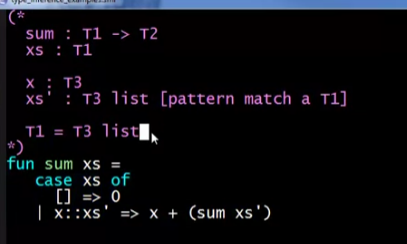
Next function: **sum**



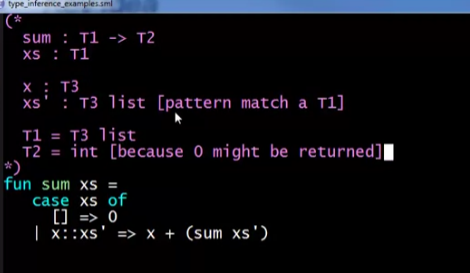
Initial type of function & argument



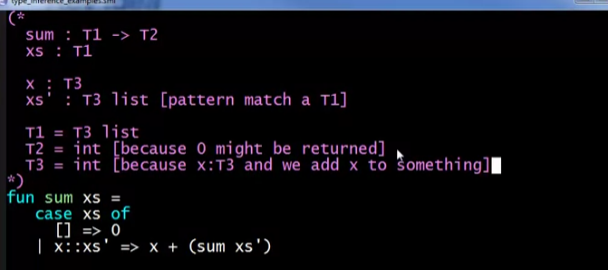
Function body



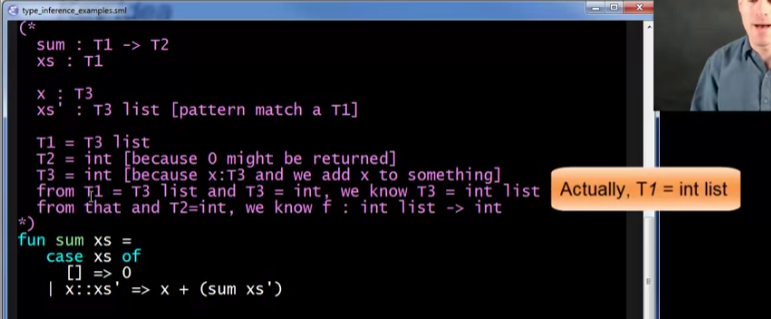
Result type T2 (based on 1st case of xs)



Looking at the 2nd case for the final type of T3

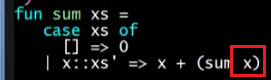


Recursive call:



Let’s try to change our function **sum**

* A mistake that will not type check
* Calling sum with the head of the list (must be rest of list because we need an int list for **sum** as an argument)



Function definition (same)

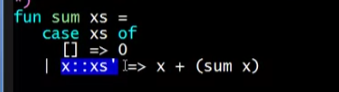


Function body (same)



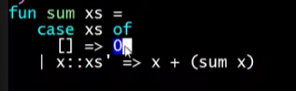
T1 (same)





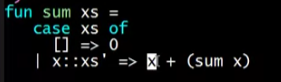
T2 (same)





T3 (same)





Recursive call



This should not type check because T1 supposed to be an int list, but x is T3 which is an int



**BUT** the type checker will not ALWAYS return an error. It will depend on the situation; it only returns an error when it reaches that part of the code. The type checker does the checking in a slightly different order. It will only tell you **one** of the type errors, not all the type errors

Try:



