(a) Under static scoping, the value of result() is 16. On line 3 in the function declaration `let f = fun y -> x + y`, x takes the value 2 because that is how it was defined in the program text a line earlier, so whenever that function is called it will substitute 2 in for x. On line 5, x takes on the value 7 because it was defined as 7 in the previous line, making this the closest enclosing scope of the text. Same with line 6, where x takes on the value 7 based on the definition in line 4. Therefore the line `x + f x` will evaluate to `7 + (fun y -> 2 + y) 7 = 7 + 9 = 16`.

(a) Under dynamic scoping, x + f x would be 21, because the most recent value for x on the stack is 7, so the line `x + f x = 7 + (fun y -> 7 + y) 7 = 7 + 7 + 7 = 21`. For line 3, x is 2, so the function f is defined `let f = fun y -> 2 + y`, but then in line 5 and 6, the most recent definition on the stack has x as 7, so line 5 is `7 +` and line 6 is `x+ (fun y -> x + y) x` and 7 is substituted in for all x's.

(a) F# uses static scope. One way I know is if we think about curried functions, and the type `'a -> ('b -> 'c)`. We can imagine a curried function of this type, `f x y`, and then imagine only passing it a value for x, meaning that it will return a function `('b -> 'c)`. We can then use this new function, and it will not be affected by associating x with a new stack frame. This implies that F# is using the closest value in the program text, not the most recent on the stack.