## Guidelines for Using set! and mutators

When should you use set!, and when should you use a mutator? Here are some guidelines:

* set! operates on a *variable* defined outside the function. It can’t be used on information that’s passed as a parameter to a function, or on information that’s the result of a computation (in other words, you need to provide set! with a variable name, not with the result of a computation).
* A mutator, by contrast, can operate on any expression that produces a structure, even if the struct is passed as a parameter or is the result of a computation.
* If you’re working with a list of structures,
	+ if the change is confined to one structure, a mutator is the better choice. Think of the deposit function we wrote for Citibank. We were able to define that function using either set! or a mutator, but using the mutator resulted in the function having to do less work.
	+ if the change pertains to the entire list, then you need to use set! An example of this is removing an account from Citibank. A mutator just changes one component of a struct; it’s not meant to be used to change the number of items in a list, as remove-account does.
* If you’re working with an individual structure,
	+ if the same struct is used in multiple places (it’s shared), and you want the change to be visible from all of those places, use a mutator. This is what we did when we had two customers, Maria and Phil, who shared a joint bank account. We used set-account-balance! on their joint account so that any changes made were visible from both customers.
	+ if you want the change to be visible from only one place, use set! to replace the structure with a new structure. This is what we did when we used set! to redefine the account MPAcct. For another example, let’s say that Maria decides to leave Phil, and goes to the bank to set up her own account. You could use set! to create a whole new customer structure for Maria, including new account information.