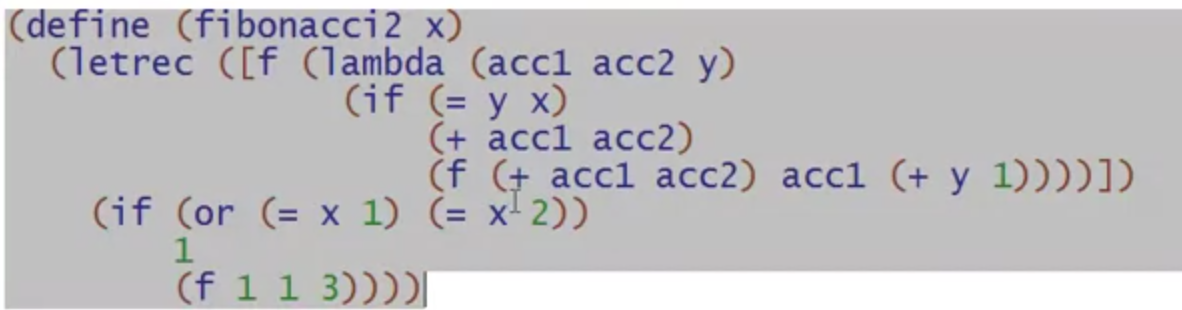
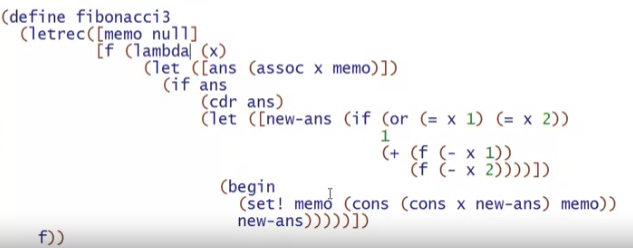


Other versions



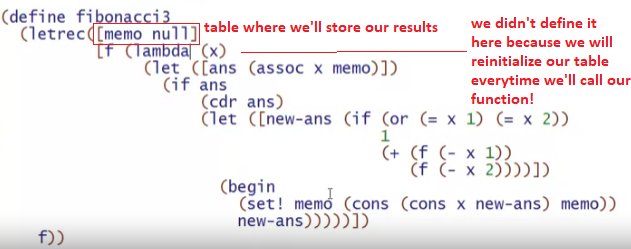
Different way to do it MORE EFFICIENTLY

* Keeping around all the previous results

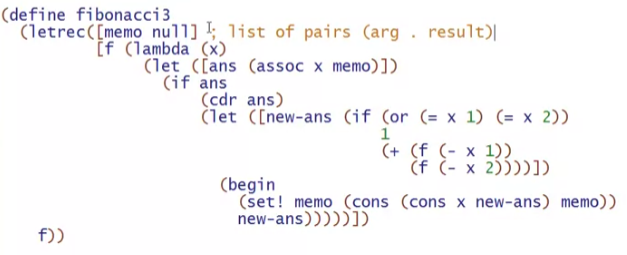


Line-by-line explanation

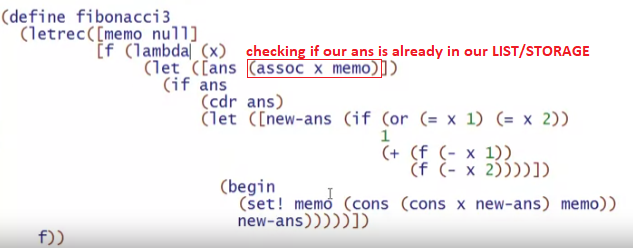
Our storage:

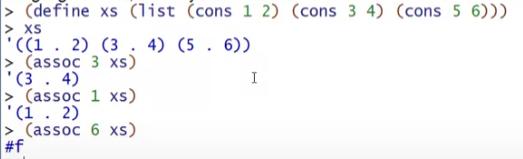


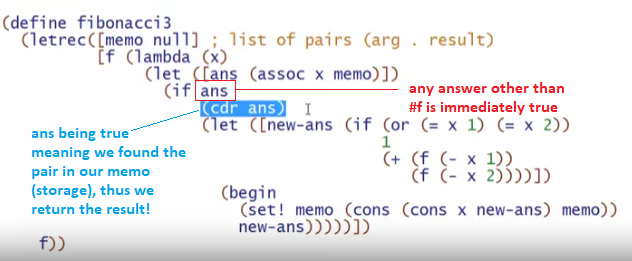
* It persists on one call and another



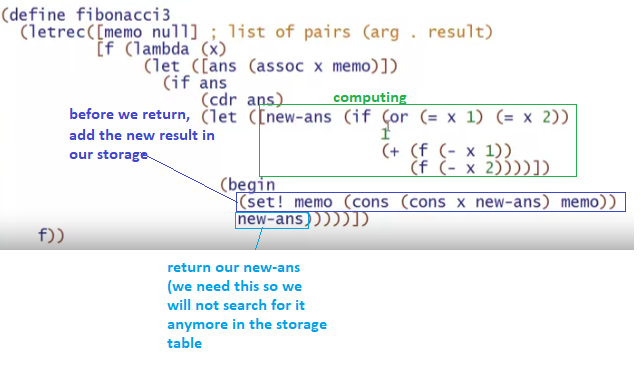
Our function:



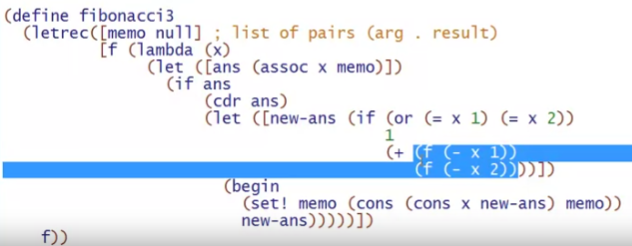
* **assoc** is a predefined function (check Racket language)
  + checks all **car** of a list and returns the pair that has that car value
  + if there are no car values that matches the first argument in assoc, returns false
  + 



If we didn’t find it, that will be the time we will compute it



Why is this fast even if we do 2 recursive calls?



* The first recursive call ends up filling our STORAGE TABLE (memo) with lots and lots of numbers
* Thus the second recursive ends up searching from our storage table and finding the result ALREADY in our table (most of the time)
* Twice as fast AT EVERY LEVEL

