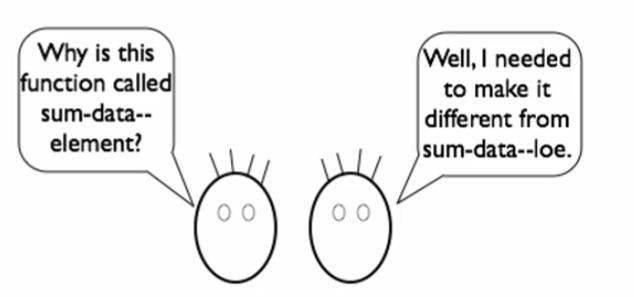
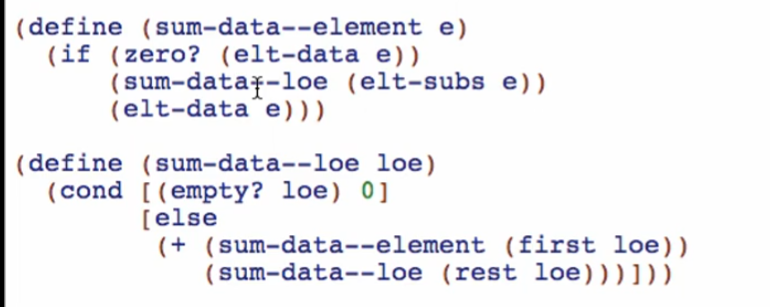
Lexical scoping is the foundation of encapsulation, a crucial technique for managing complexity of large programs.

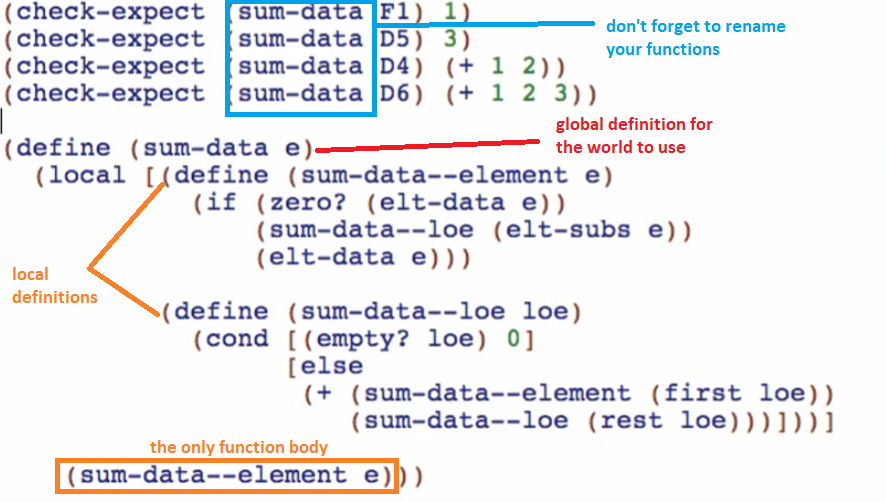
Encapsulation

* Make our program a little package that has a bunch of internal functions, constants, structures that is only true or only exists within that package



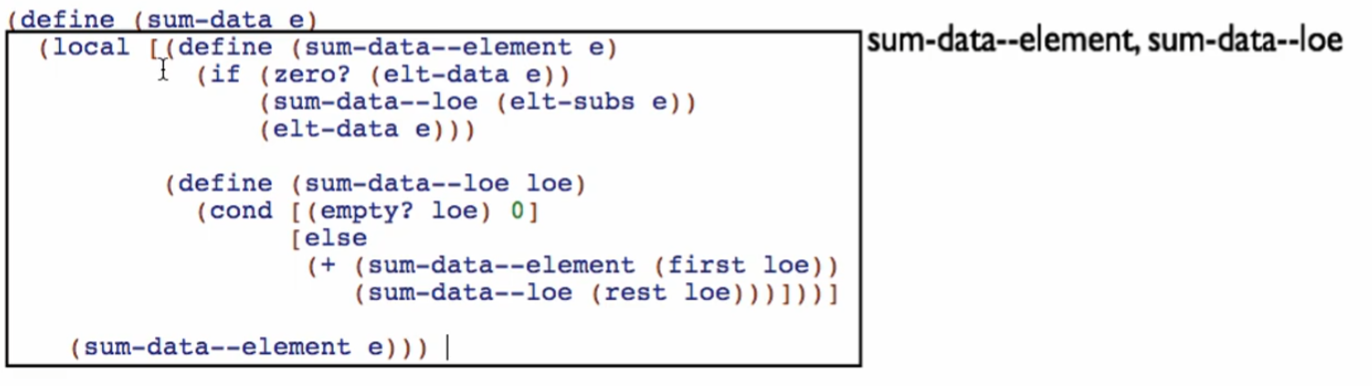


How to fix it using local: (using the same name but different functions)



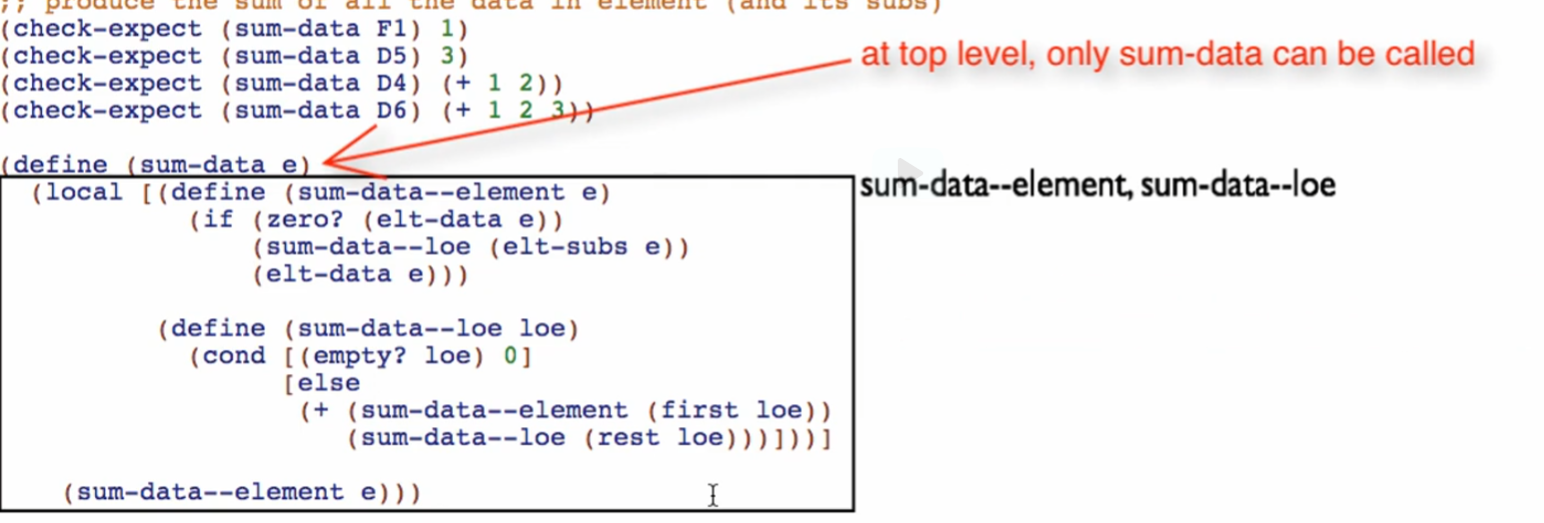
Run and check: 

The two functions are now encapsulated



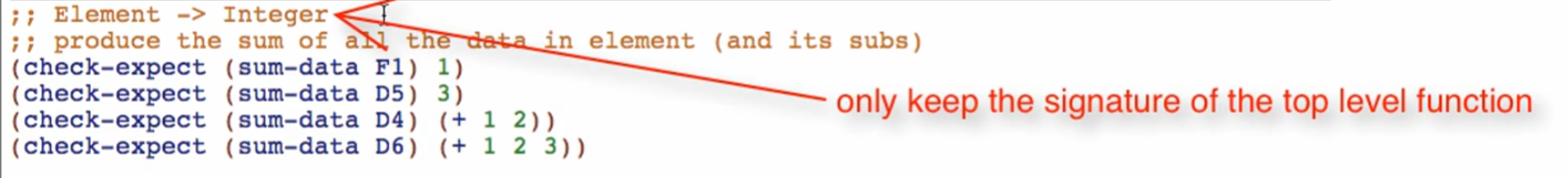
* Nobody outside this local can access these functions

The only function for top level/global is sum-data



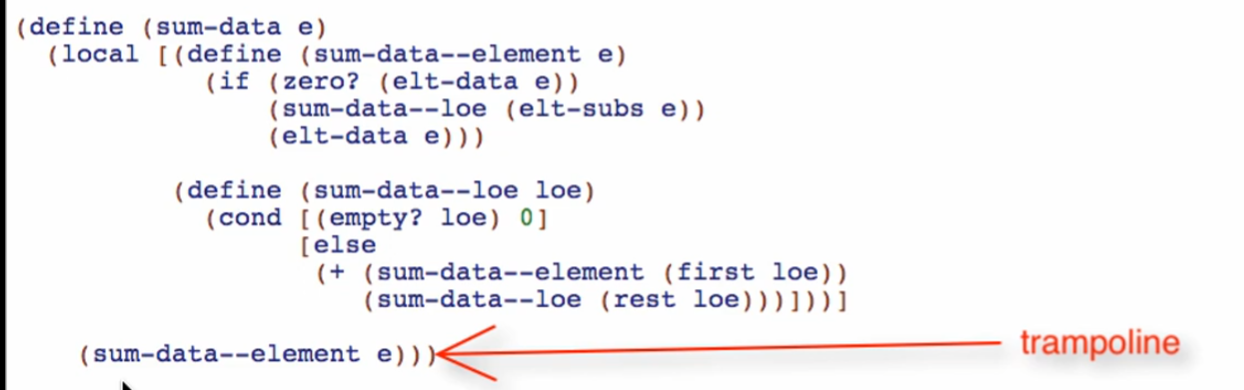
* sum-data ENCAPSULATES the 2 functions

Because of that, we will update the signature, 1 function only and check-expects





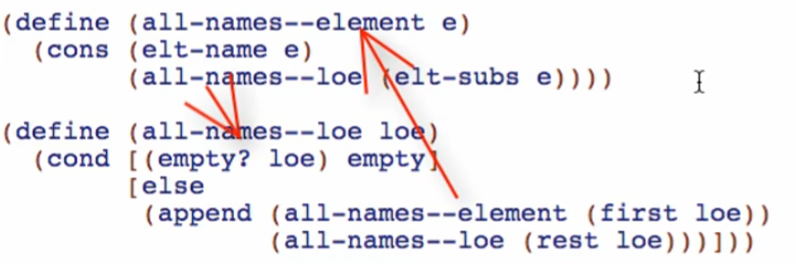
* We only test elements just to keep check-expects simple in Dr Racket
* But in other programming languages, you can test sum-data—element and sum-data—loe!



* Because we are bouncing off from element -> list of elements -> element … until the recursive function is done

How to know if functions are good for encapsulation

* 2 or more functions work together



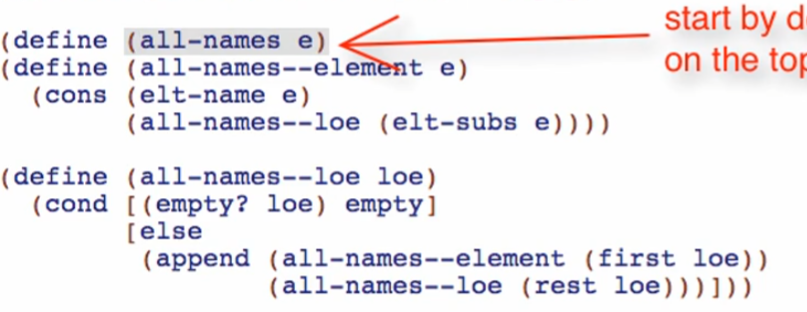
* But the rest of the program only wants to call 1 function

Good candidates for encapsulation:

1. One function has one or more helpers closely linked to it.
2. The outside program really only wants to call the **main** function, not the helpers.

Another example:

Start by defining the function you want to be seen/called on the top level



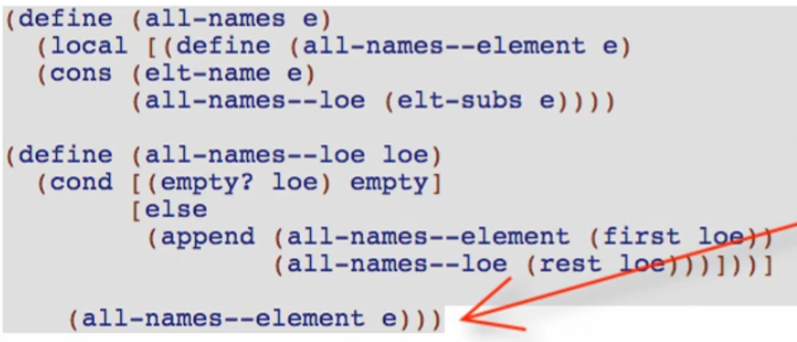
Next is starting the local expression

Enclose all definitions using [ ] brackets



Write the trampoline (Code Body)

* Calling the appropriate function in the local definitions with the argument passed from the outer function



Enclose and fix indentation



Next steps

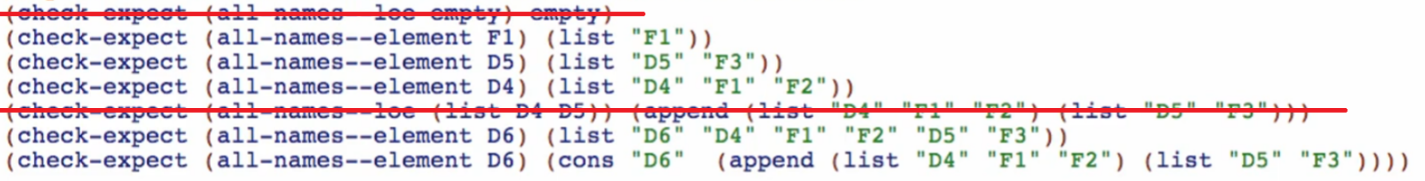
Keep only the signature of the main function (all-names)



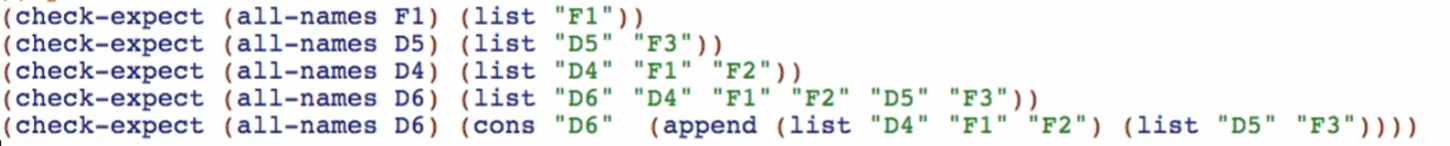
Clean up the tests

* Often times we get rid of the base case tests

Get rid of the test for the now hidden functions



Rename all the main function



Clean up the stub



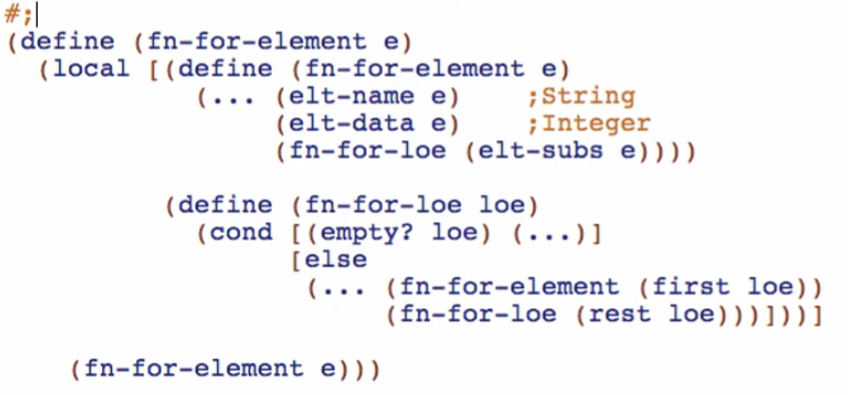
Run and test!

This follows the idea of REFACTORING

* Changing a program’s code/structure without changing the program’s behavior at all!
* Do not CHANGE any behavior WHILE REFACTORING
  + First do the refactoring,
  + Check and test if the current behavior is correct
  + And then change the behavior

For functions with mutual references, we can pre-encapsulate the templates:

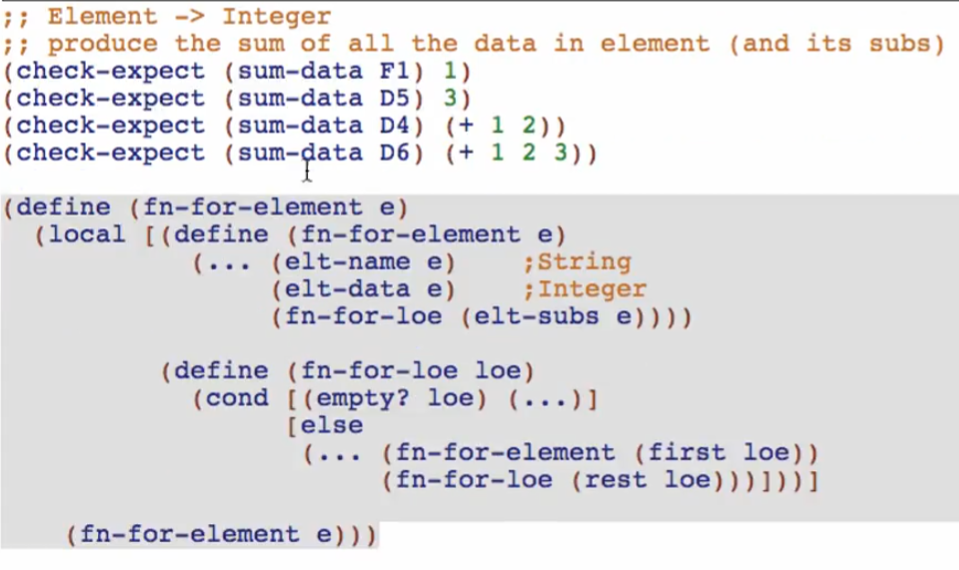




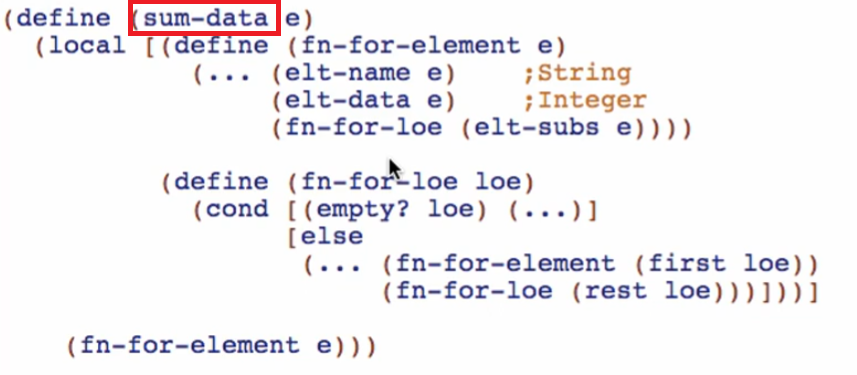
* We can do this but the following would be the consequences

**Consequences of pre-encapsulating the template of mutually referenced functions**

Copy template and try to paste it in the function body

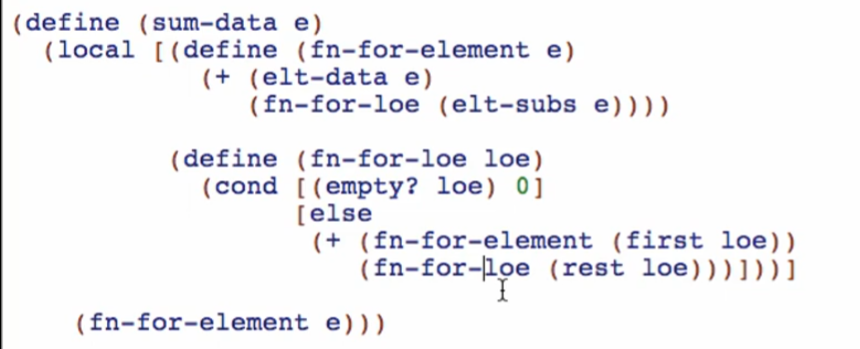


Then we rename the template

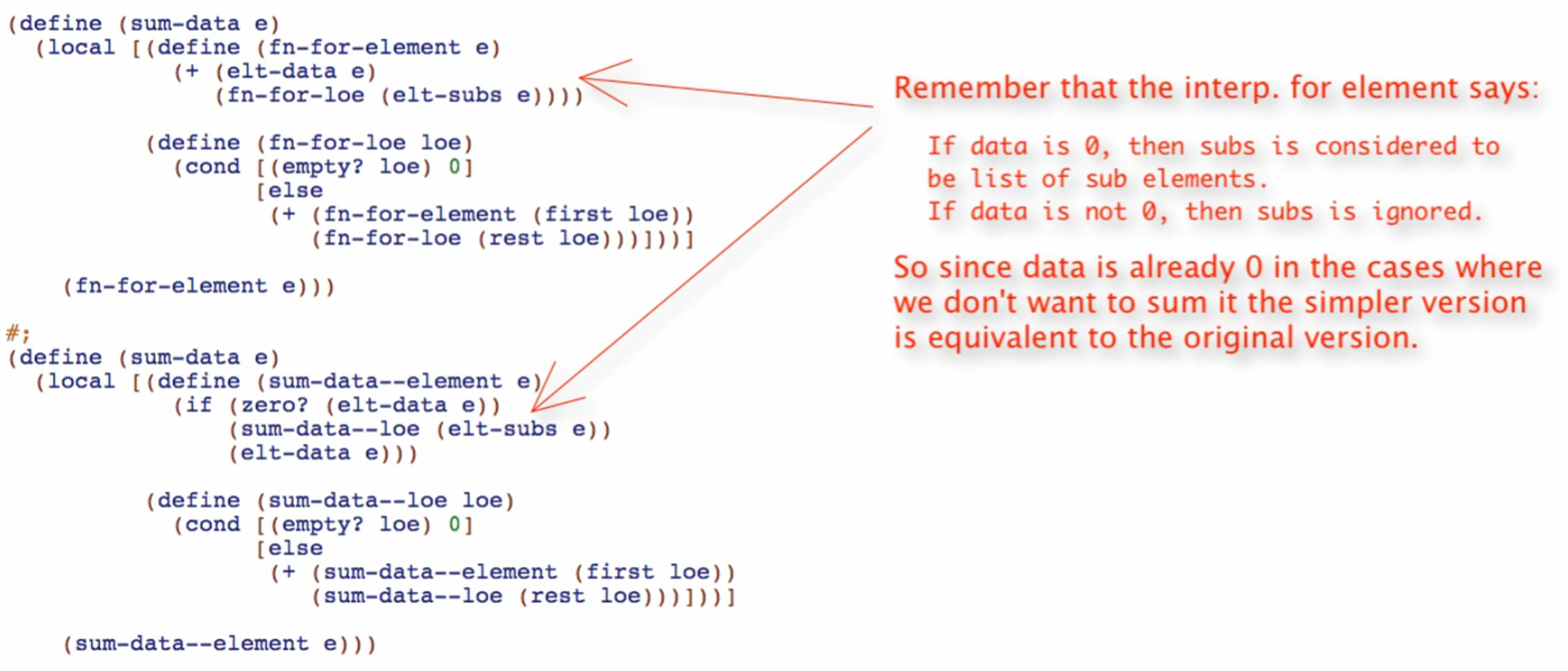


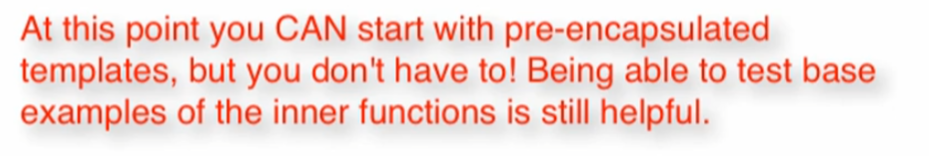
And we don’t have to bother to rename the internal/local functions

We then code the function body



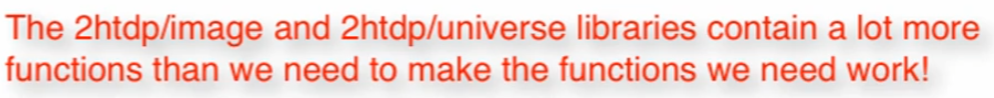
We can now run it and we’ll see that it passed



Big disadvantage of this: **NO BASE CASE** TEST DURING DESIGN PROCESS****

Encapsulation

* Very vital when multiple programmmers work on huge programs
* It also helps in eliminating the risk of calling a function you don’t suppose to call



* These are examples of 2 encapsulations!

Local

* Allows us to implement encapsulation in ISL!