INTRODUCTION TO

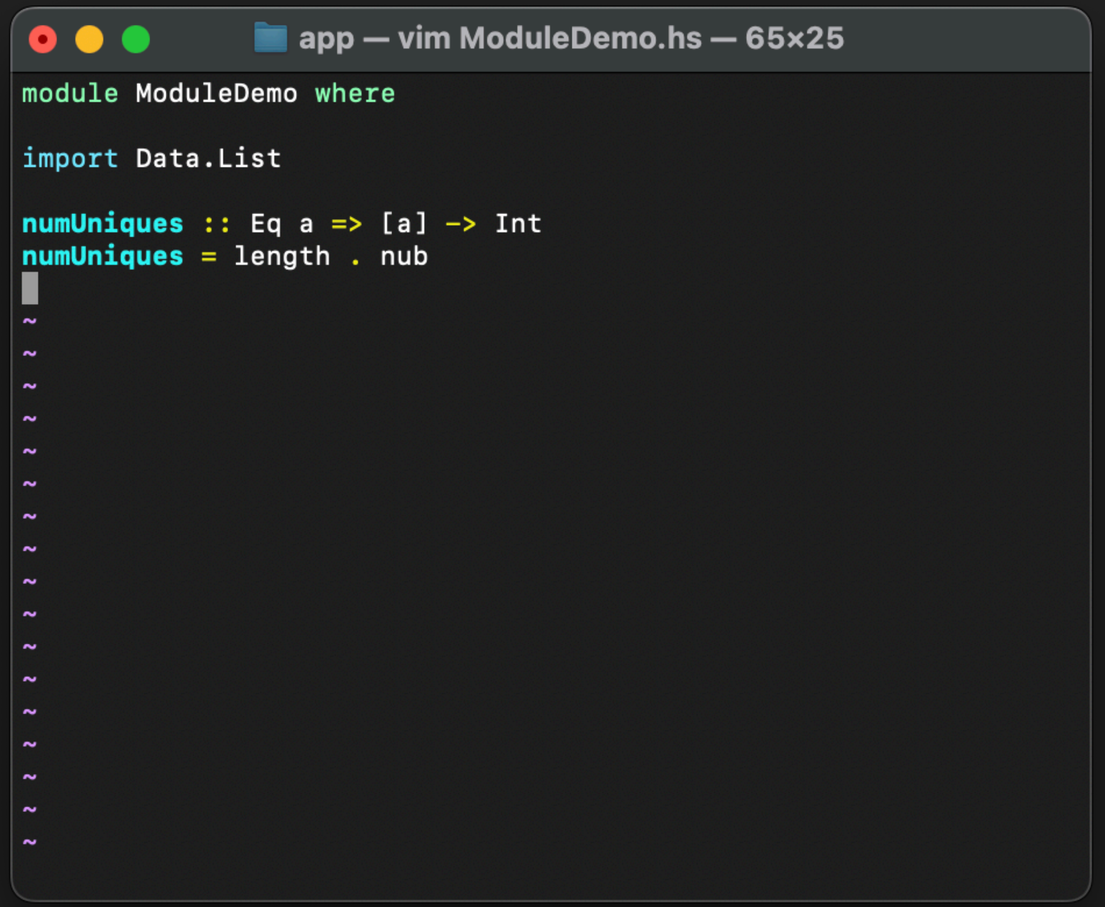
FUNCTIONAL PROGRAMMING

OUTLINE

- Modules
- Packages
- Creating your own package
- Unit testing

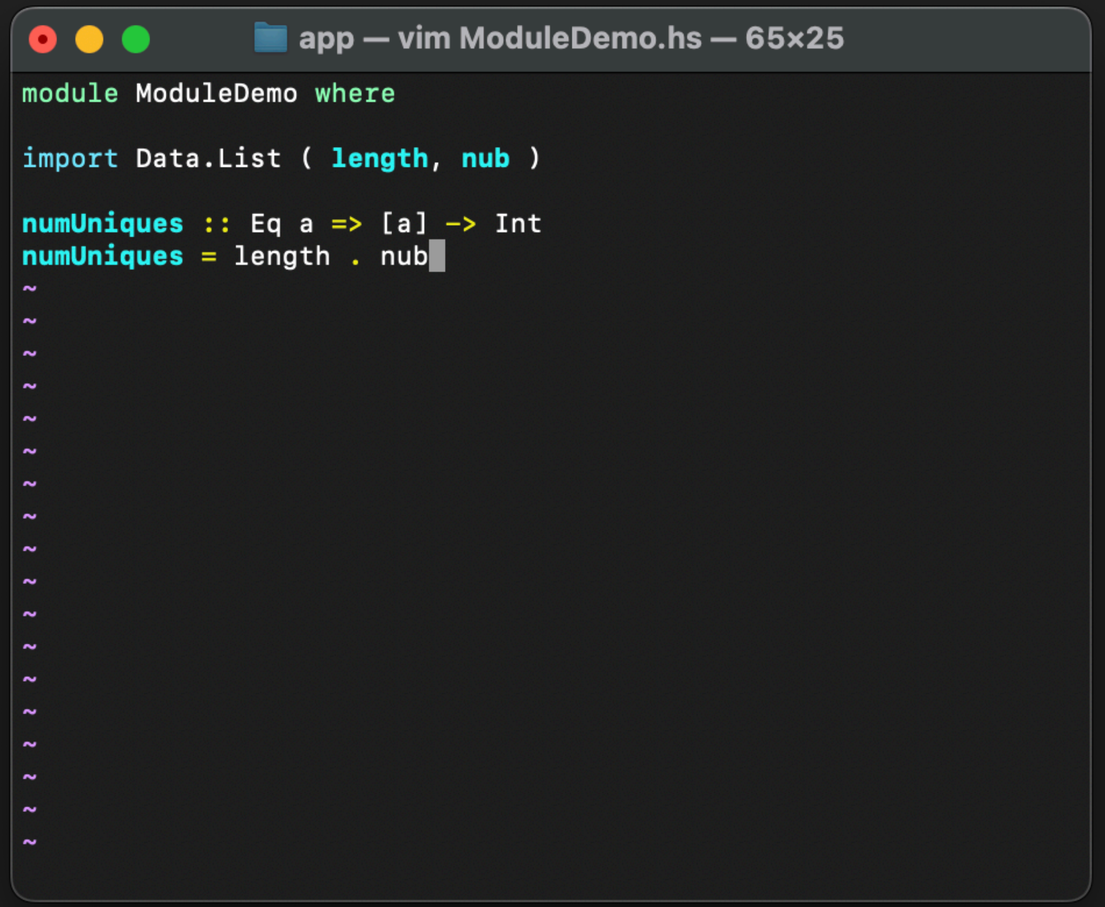
MODULE SYSTEM: IMPORT

- Module is a collection of related functions, types, and typeclasses
- Use import moduleName to import everything
- Add (f, g) to import only f and g
- Hide some names with hiding
- Use fully qualified names with qualified



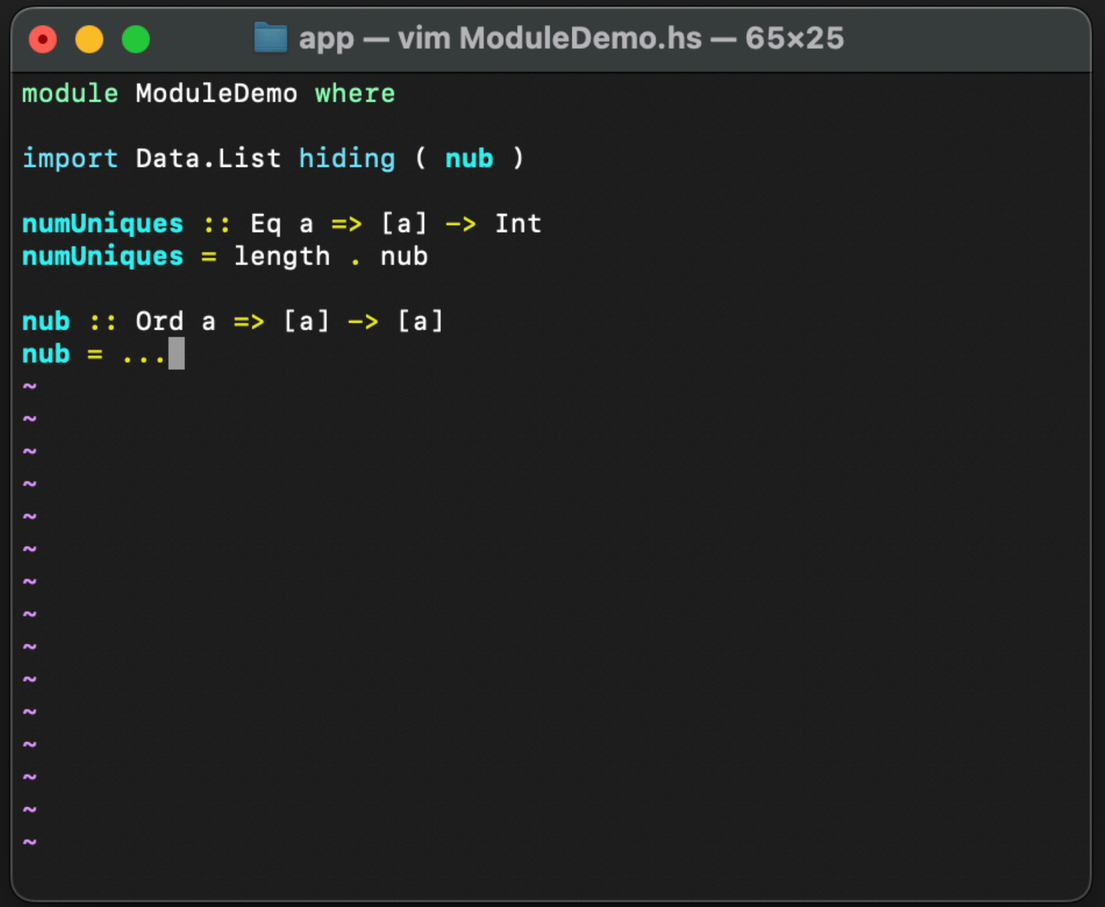
MODULE SYSTEM: IMPORT

- Module is a collection of related functions, types, and typeclasses
- Use import moduleName to import everything
- Add (f, g) to import only f and g
- Hide some names with hiding
- Use fully qualified names with qualified



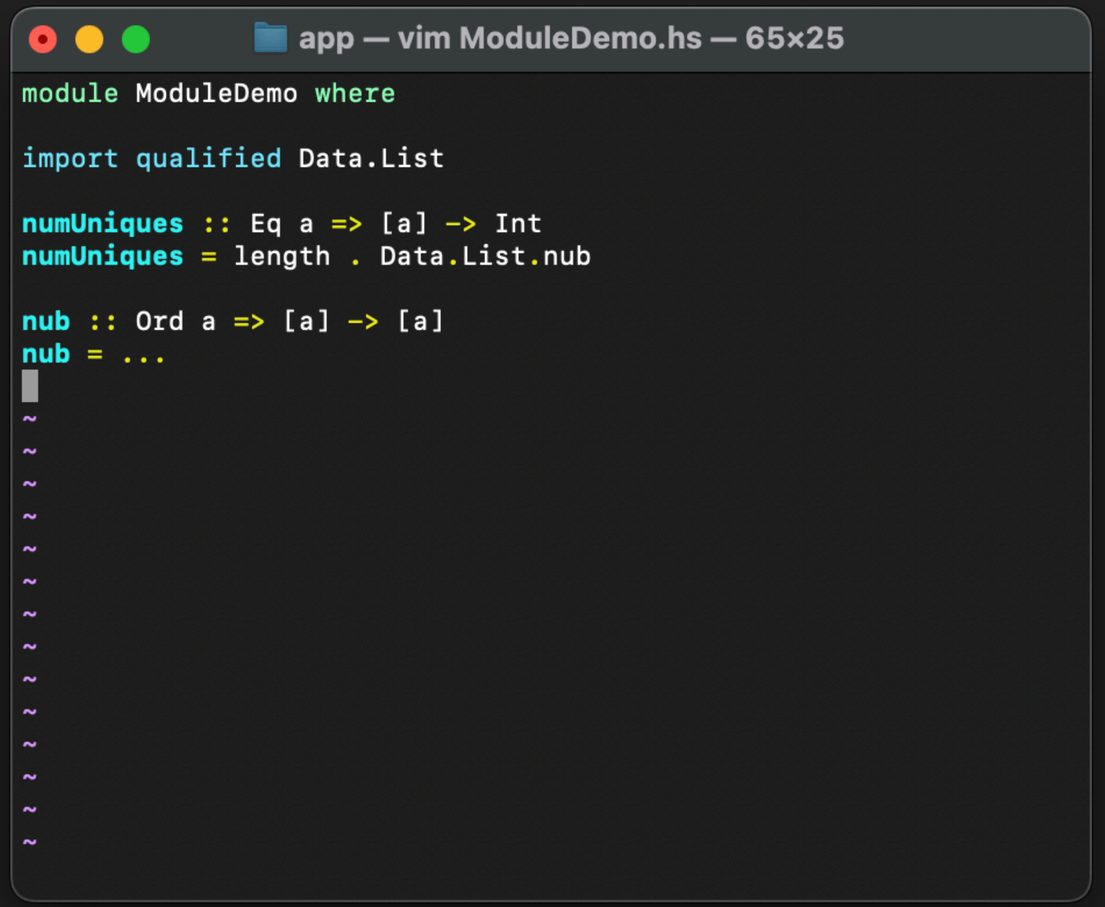
MODULE SYSTEM: IMPORT

- Module is a collection of related functions, types, and typeclasses
- Use import moduleName to import everything
- Add (f, g) to import only f and g
- Hide some names with hiding
- Use fully qualified names with qualified



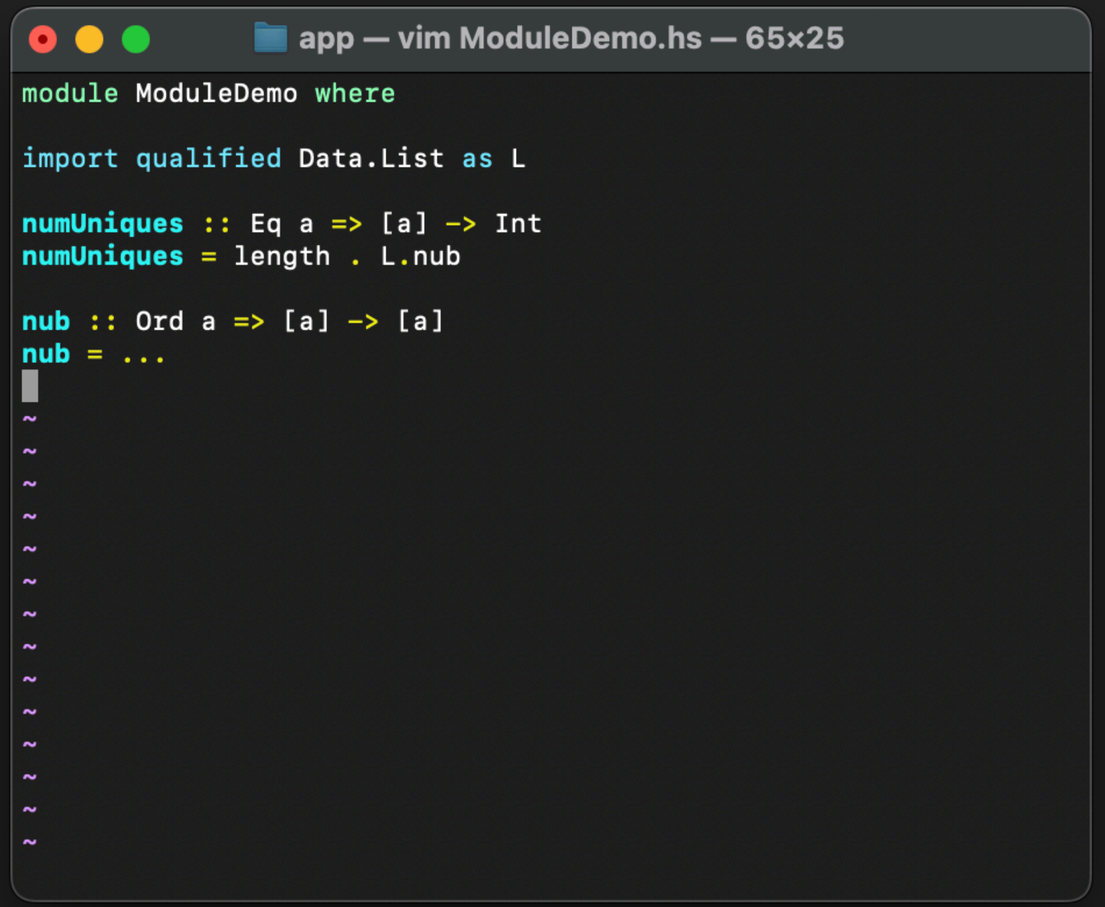
MODULE SYSTEM: IMPORT

- Module is a collection of related functions, types, and typeclasses
- Use import moduleName to import everything
- Add (f, g) to import only f and g
- Hide some names with hiding
- Use fully qualified names with qualified



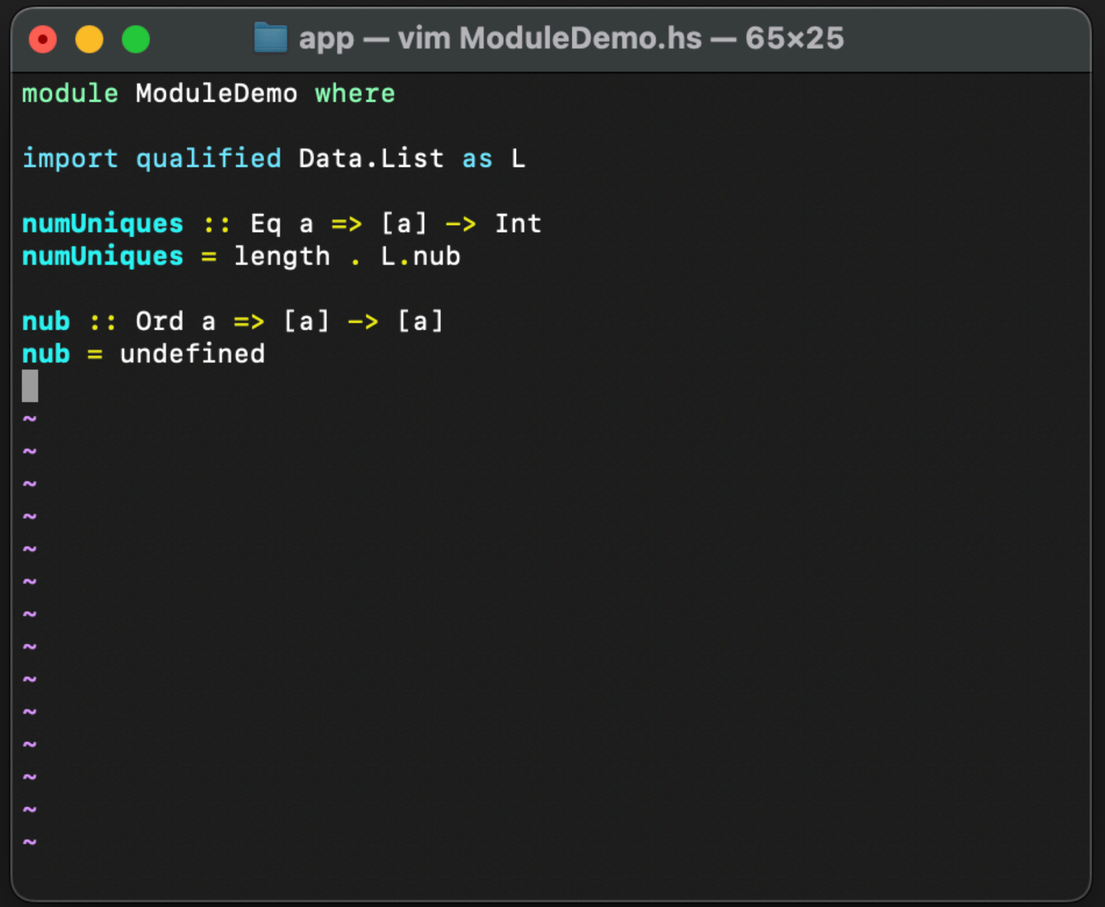
MODULE SYSTEM: IMPORT

- Module is a collection of related functions, types, and typeclasses
- Use import moduleName to import everything
- Add (f, g) to import only f and g
- Hide some names with hiding
- Use fully qualified names with qualified



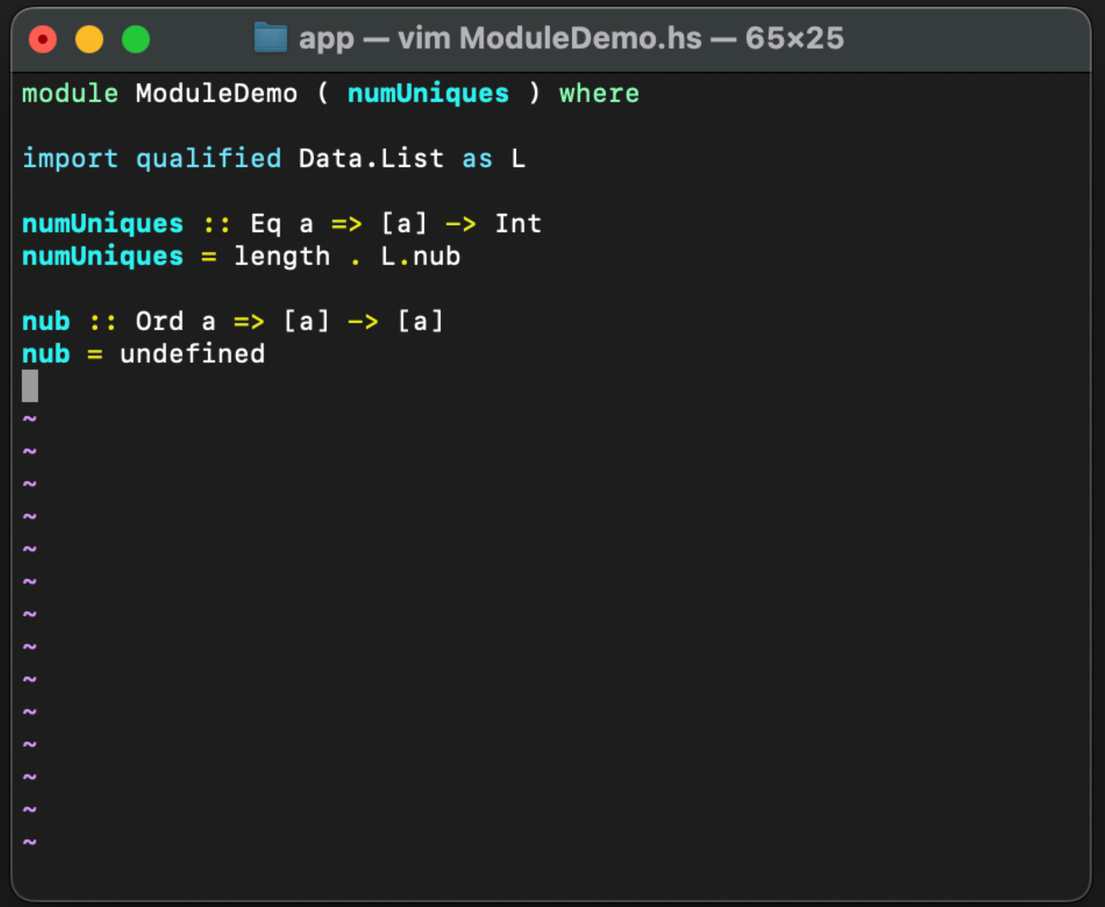
MODULE SYSTEM: EXPORT

- Everything on the top level of the module is exported by default
- You can add names in parentheses to export only them
- You can re-export names from the imported modules
  - By default, only names defined in the module are exported



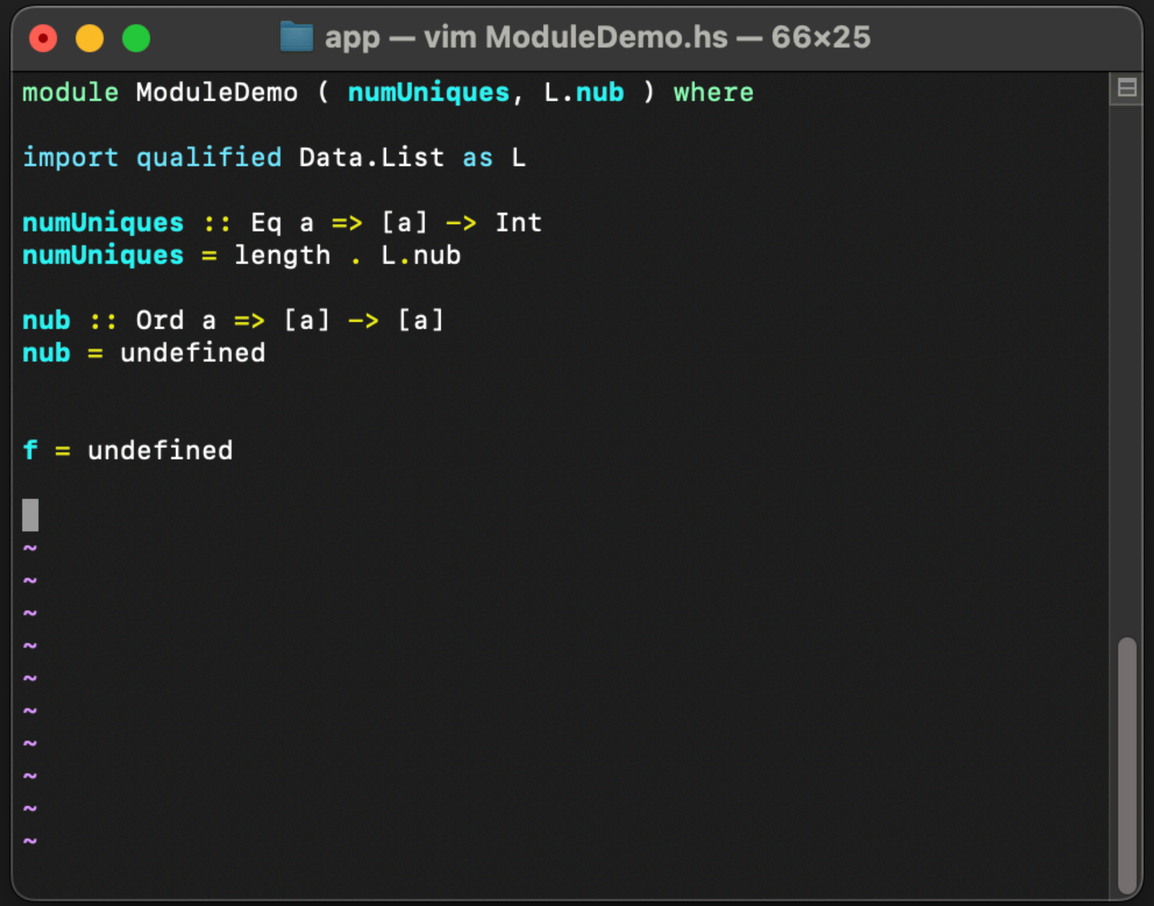
MODULE SYSTEM: EXPORT

- Everything on the top level of the module is exported by default
- You can add names in parentheses to export only them
- You can re-export names from the imported modules
  - By default, only names defined in the module are exported



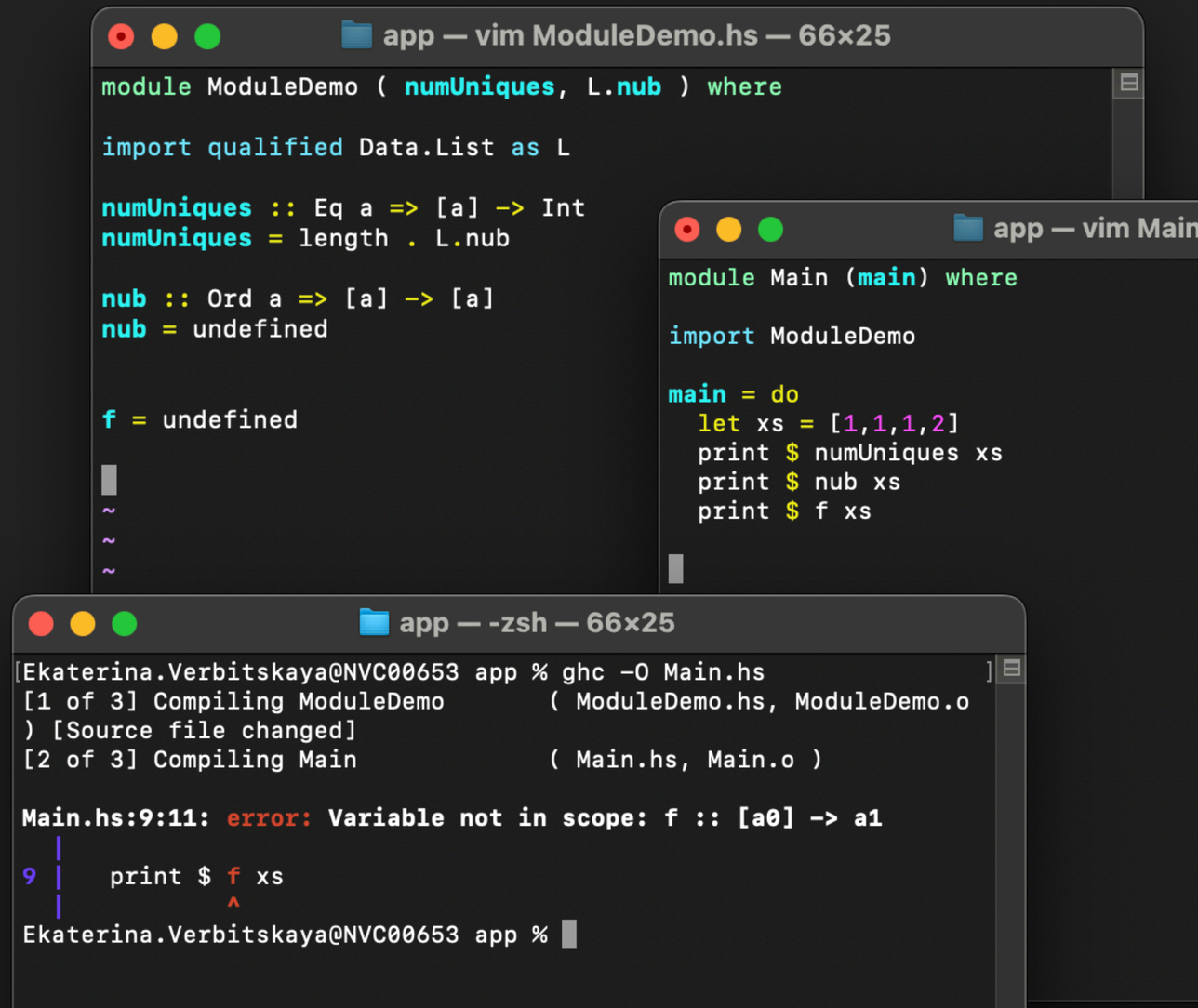
MODULE SYSTEM: EXPORT

- Everything on the top level of the module is exported by default
- You can add names in parentheses to export only them
- You can re-export names from the imported modules
  - By default, only names defined in the module are exported



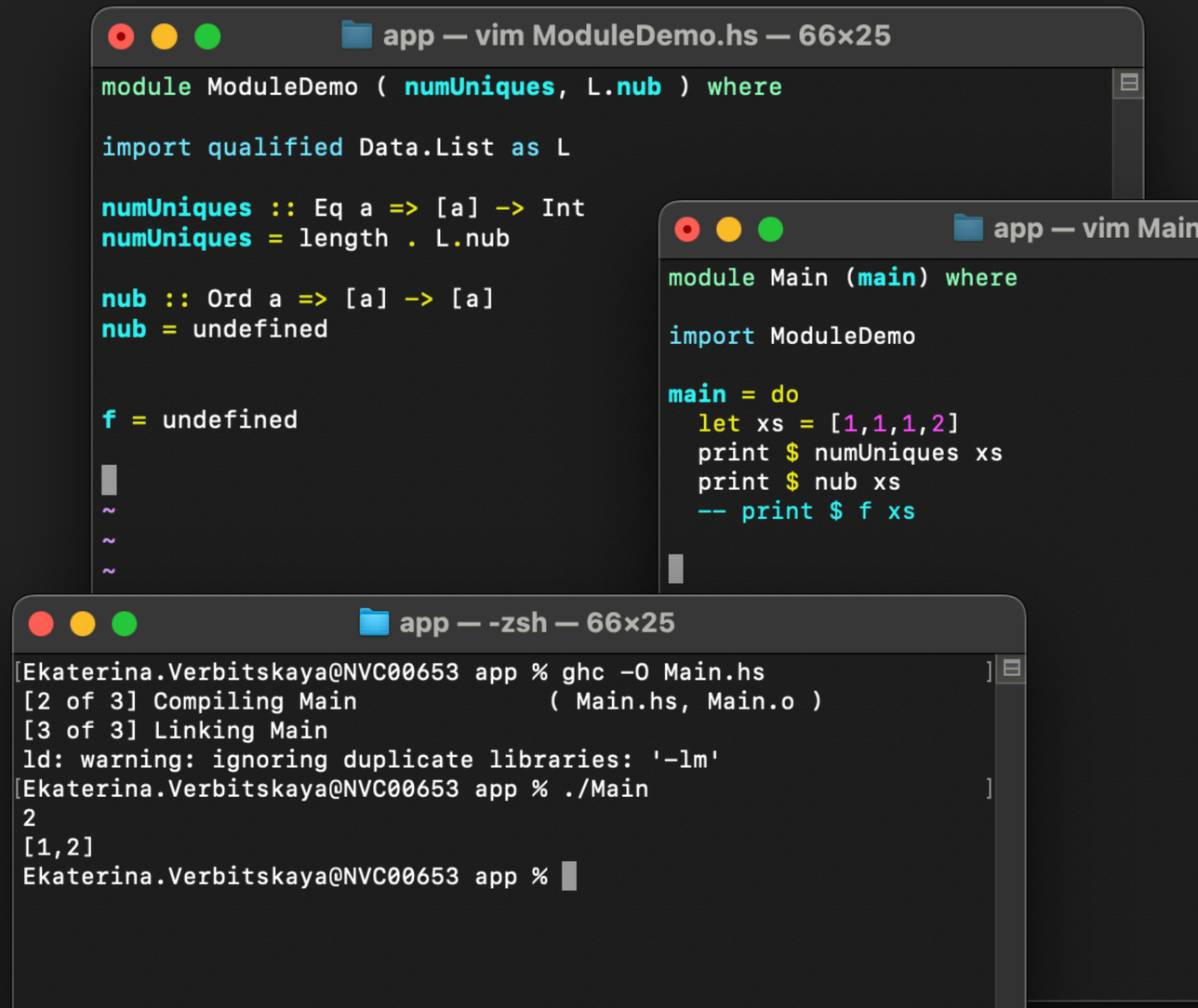
MODULE SYSTEM: EXPORT

- Everything on the top level of the module is exported by default
- You can add names in parentheses to export only them
- You can re-export names from the imported modules
  - By default, only names defined in the module are exported



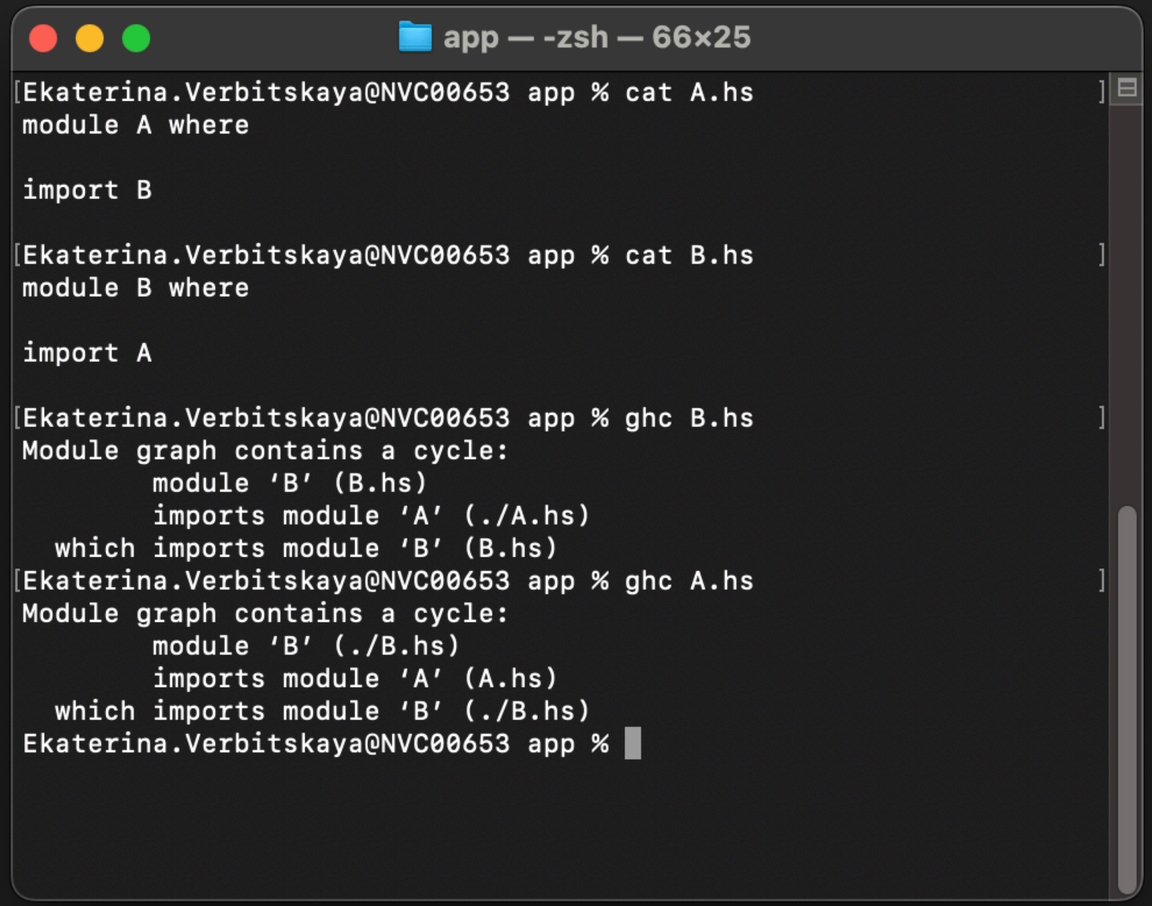
MODULE SYSTEM: EXPORT

- Everything on the top level of the module is exported by default
- You can add names in parentheses to export only them
- You can re-export names from the imported modules
  - By default, only names defined in the module are exported



MODULE SYSTEM: CYCLIC DEPENDENCIES

- Cyclic dependencies are not allowed



EXERCISE

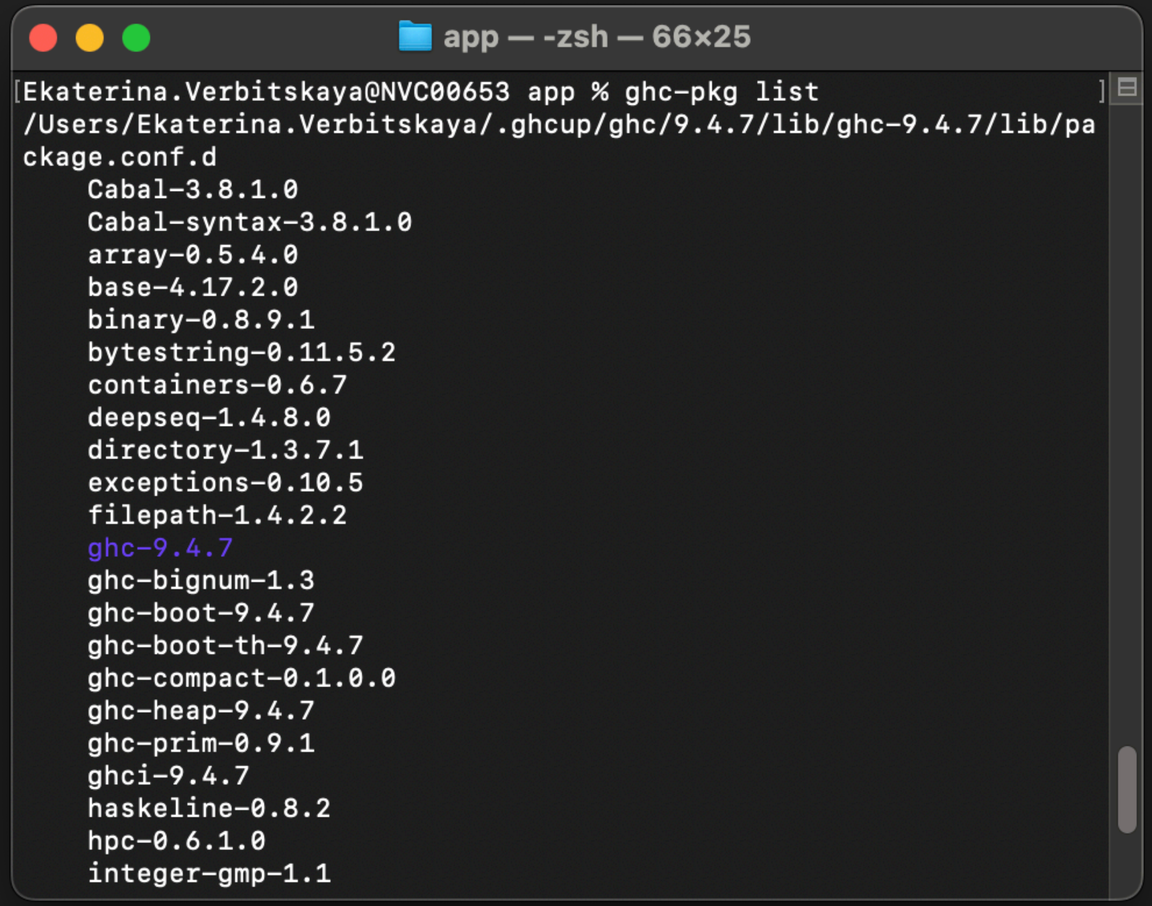
- Make three modules A, B, and C
- Add functions with names f, g, and h into each of the modules
- Make C depend on both A and B
- Make B depend on A
- Make C export only its function h and re-export the function f of module A

OUTLINE

- Modules
- Packages
- Creating your own package
- Unit testing

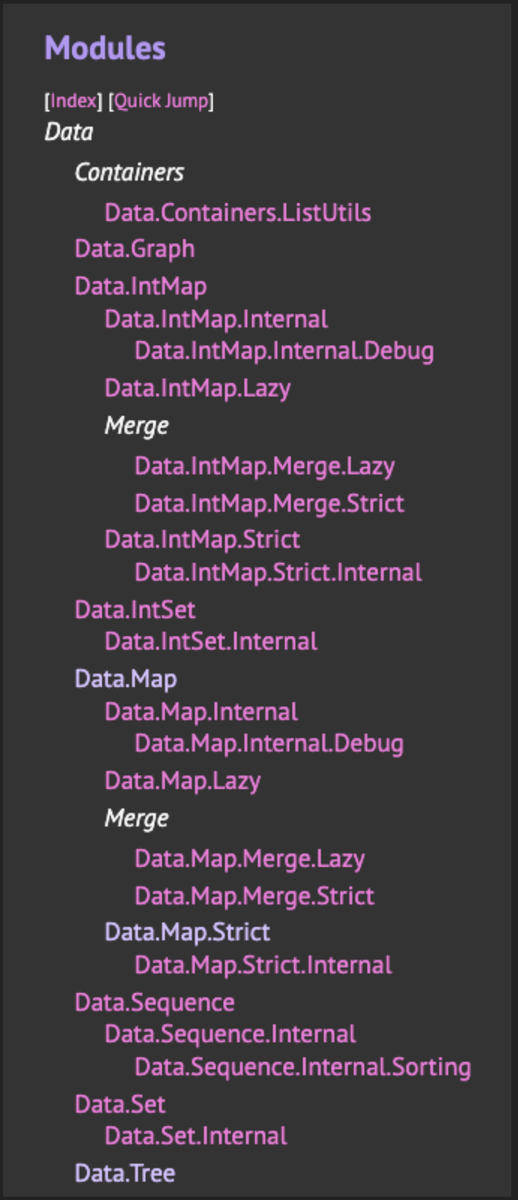
WHAT ARE THEY?

- Packages are collections of libraries
- Packages are units of distribution
- There might be some packages already installed on your system
- If the package you need is not installed, you can do it by stack install or cabal install



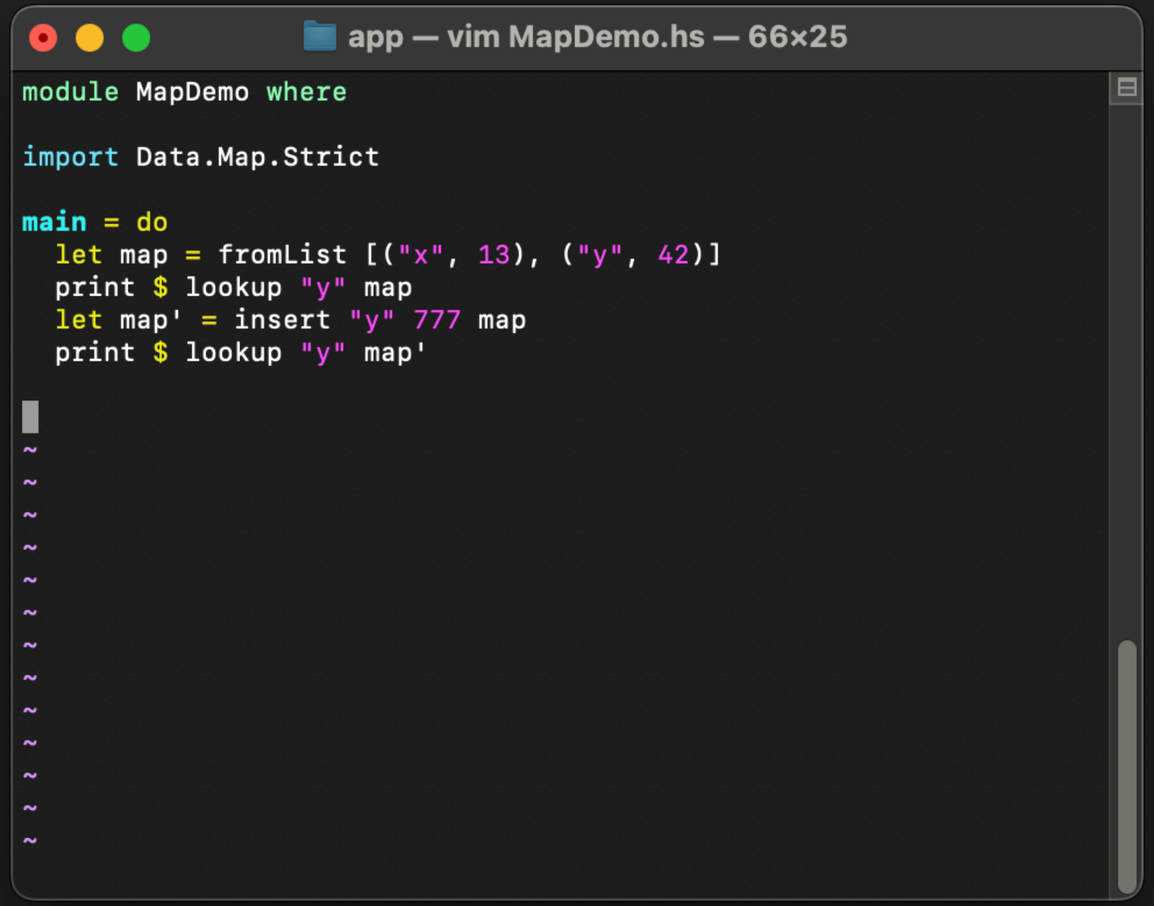
PACKAGE CONTAINERS

- A container is a data structure which holds some data, such as a dictionary or a tree
- Data.Map, Data.Set, Data.Tree…
- Many containers come in strict and lazy versions
  - Use the strict version if you need to access all of the values eventually
- docs



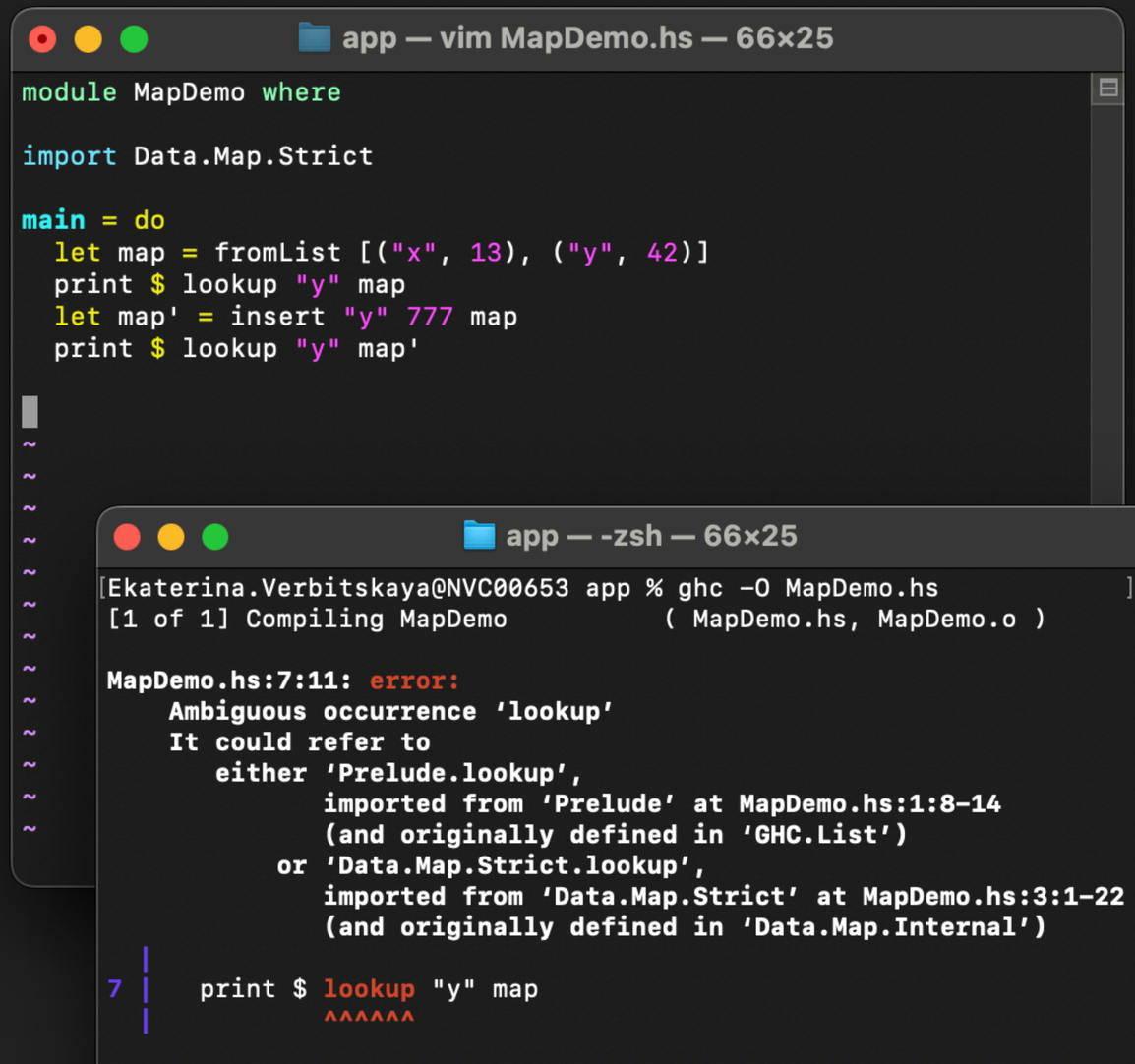
CONTAINER MAP

- Map k v is a finite dictionary with keys of type k and values of type v



CONTAINER MAP

- Map k v is a finite dictionary with keys of type k and values of type v
- Ambiguous occurrence means that the same name comes from different modules



CONTAINER MAP

- Map k v is a finite dictionary with keys of type k and values of type v
- Ambiguous occurrence means that the same name comes from different modules
- Use qualified imports!

app — vim MapDemo.hs — 66×25

```
module MapDemo where

import qualified Data.Map.Strict as M

main = do
  let map = M.fromList [("x", 13), ("y", 42)]
  print $ M.lookup "y" map
  let map' = M.insert "y" 777 map
  print $ M.lookup "y" map'
```

app — ghc-9.4.7 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc...

```
[Ekaterina.Verbitskaya@NVC00653 app % ghci MapDemo.hs
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling MapDemo          ( MapDemo.hs, interpreted )
Ok, one module loaded.
[ghci> main
Just 42
Just 777
ghci>
```

EXERCISE

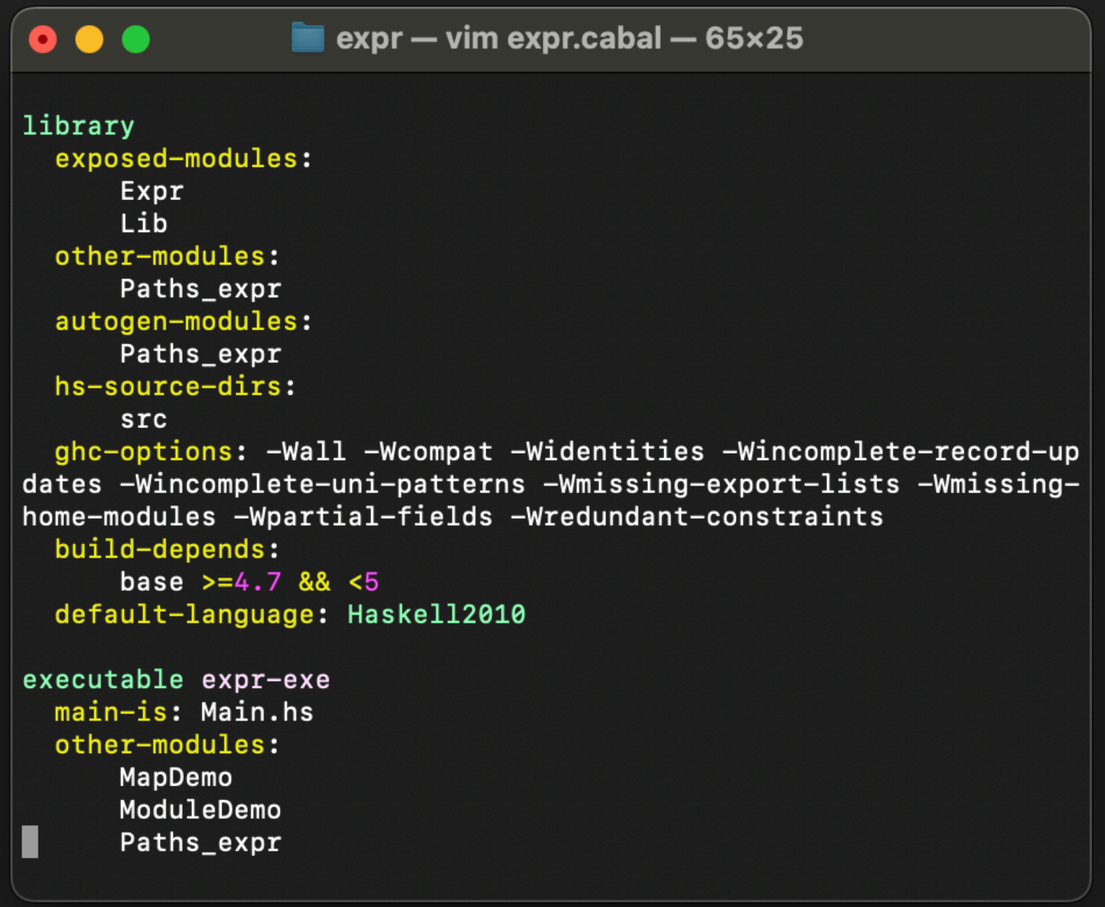
- Replace associative list with a dictionary from Data.Map in eval in HW04

OUTLINE

- Modules
- Packages
- Creating your own package
- Unit testing

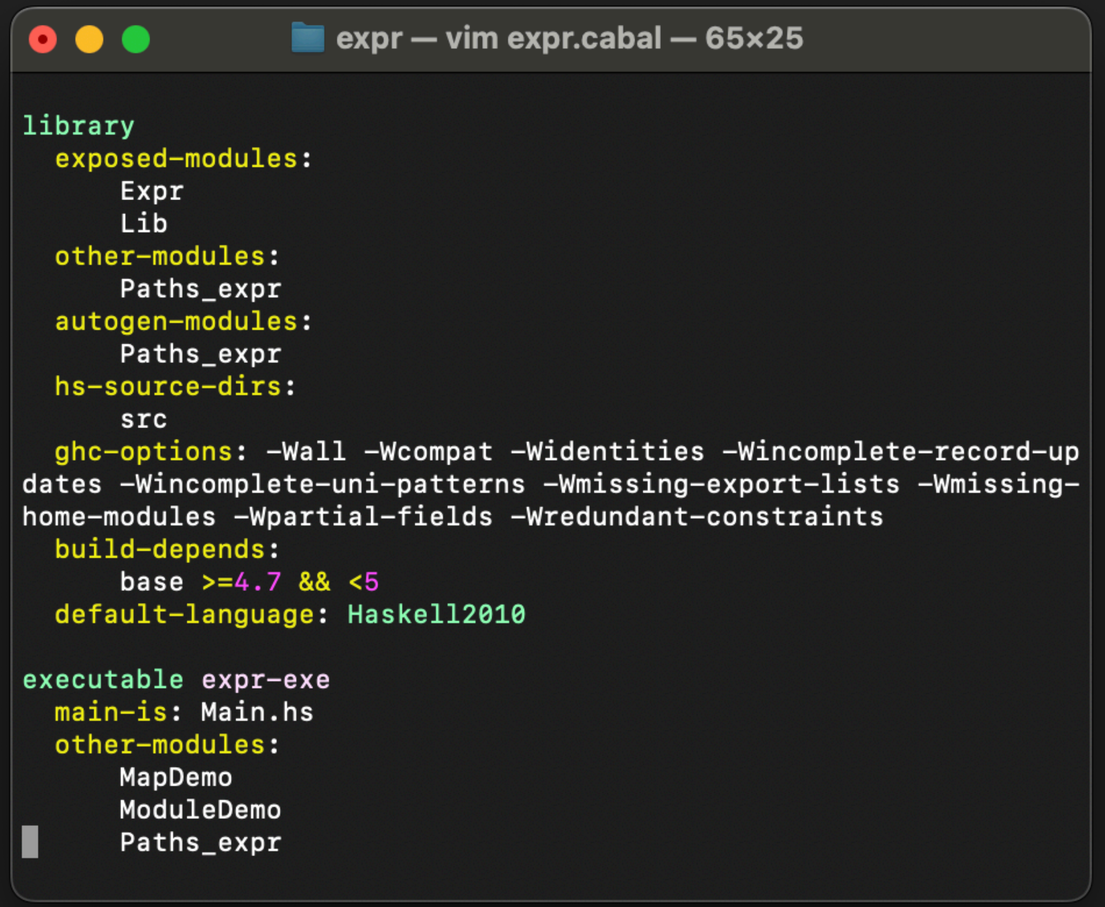
CABAL

- Build system for Haskell project
- Resolves dependencies specified in *.cabal file
- Builds libs, executables, and test suits
- May need some help with resolving dependencies



STACK

- Does what cabal does, but also:
  - Sandboxes everything, including ghc
  - Guarantees no conflict between dependencies when you use Stackage lts
- I recommend using stack
- Livecoding



EXERCISE

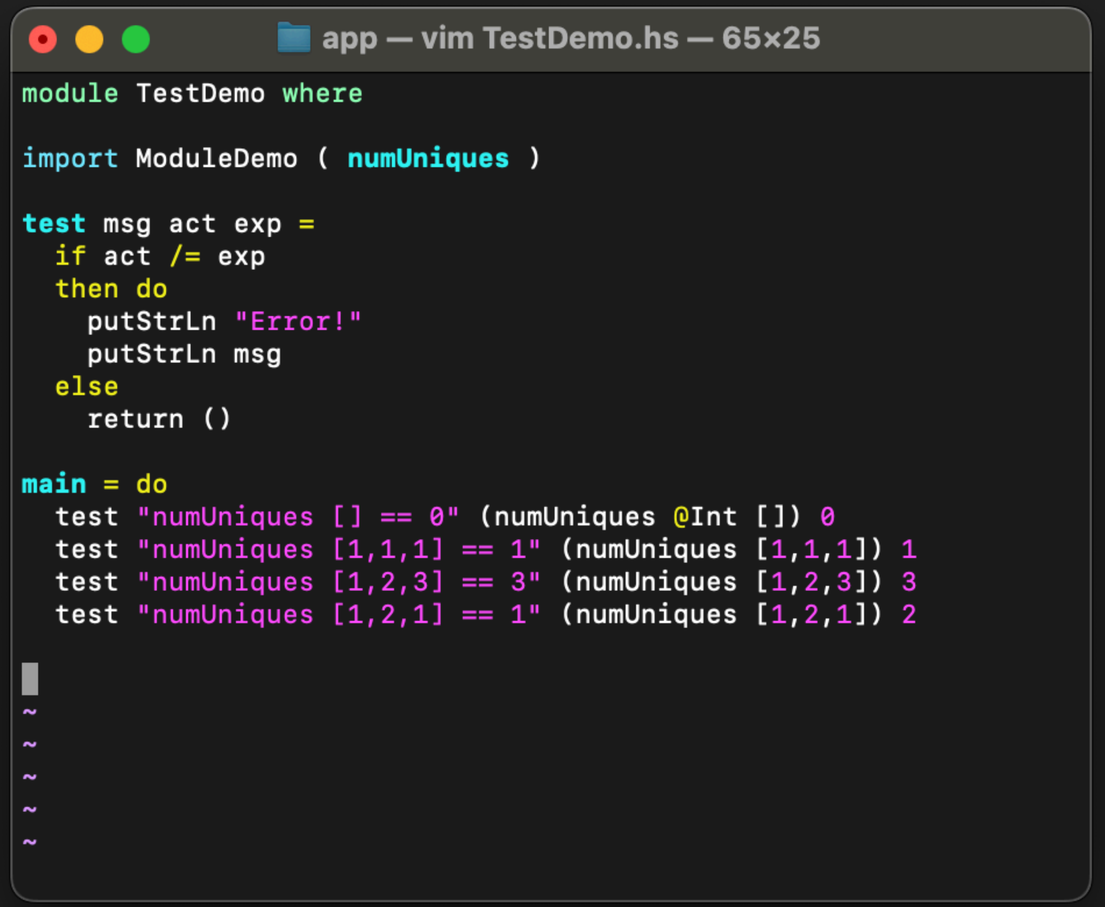
- Create a stack project
- Copy your code for Expressions there
- Make sure it builds and executes

OUTLINE

- Modules
- Packages
- Creating your own package
- Unit testing

WHAT IS A UNIT TEST

- A test which test 1 unit of functionality
- We usually test functions
  - Assert that an value computed by the function is equal to the expected
  - Assert that some predicate holds (e.g. isJust)
- Livecoding



EXERCISE

- Move your tests for Expr into a test project
- Make sure they run