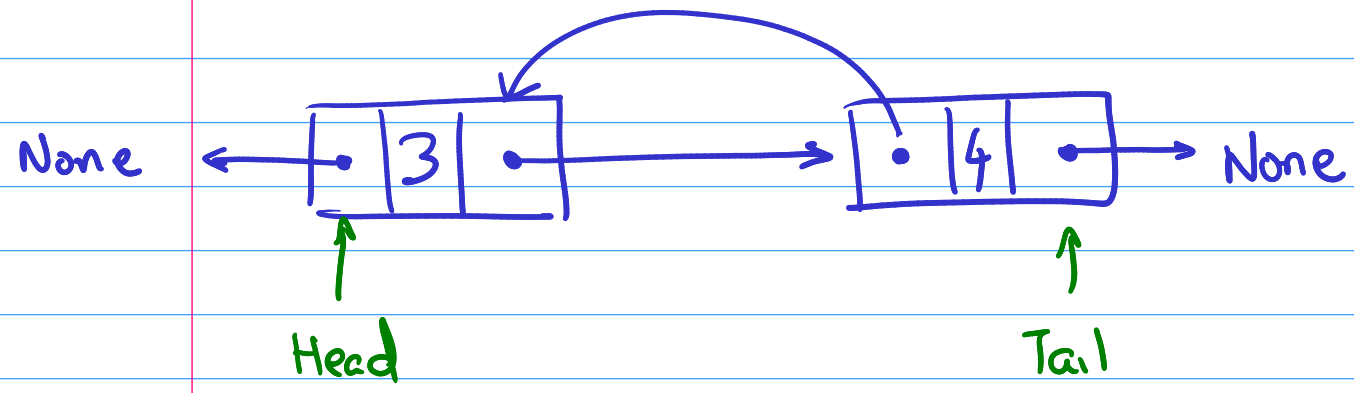
- Mutable Data Structures



- Defining recursive functions without rec!

```
let rec isEven n = if n = 0 then true else not (isEven (n - 1))
```

```
let isEven n =
  let ans = ref true in
  for i = 1 to n do
    ans := not !ans
  done;
  !ans
```

```
let helper = ref (fun x -> true)
let isEven2 n = if n = 0 then true else not (!helper (n - 1))

helper := isEven2
```

Ties the knot!

Now isEven2 works correctly everywhere!

This function returns the correct value only sometimes!

- At n=0, intentionally
- At n odd, incidentally

Cycles in space cause cycles in time.

Equality

Structural equality	Physical equality	Something else?
`'a -> 'a -> bool`	`'a -> 'a -> bool`	
`(=)`	`(==)`	
`(<>)`	`(!=)`	
Checks if the two values have the same "shape"	Checks if values point to the same memory	

Rational Numbers $\frac{3}{4}$ $\frac{1}{2}$ $\frac{2}{4}$

Numerator

Denominator

Intentional equality

Checks if underlying object is the same thing.

Extensional equality

Checks if representations are equal

⇑

Structural equality

① For basic data types, check directly

② For ADTs, confirm that top-level ctor is the same; recursively check parameters

---

Function equality: "Undecidable" in general

No algorithm exists!

Structural equality of functions is undefined

Programming in the "Large"

Q: How do we organize code?

Splitting into files / physical organizations

Functions, classes, Packages

Sense 1: Physical, syntactic organization

Split code into files, directories

"Compilation units"

#include, #use, ...

Sense 2: Organizing names

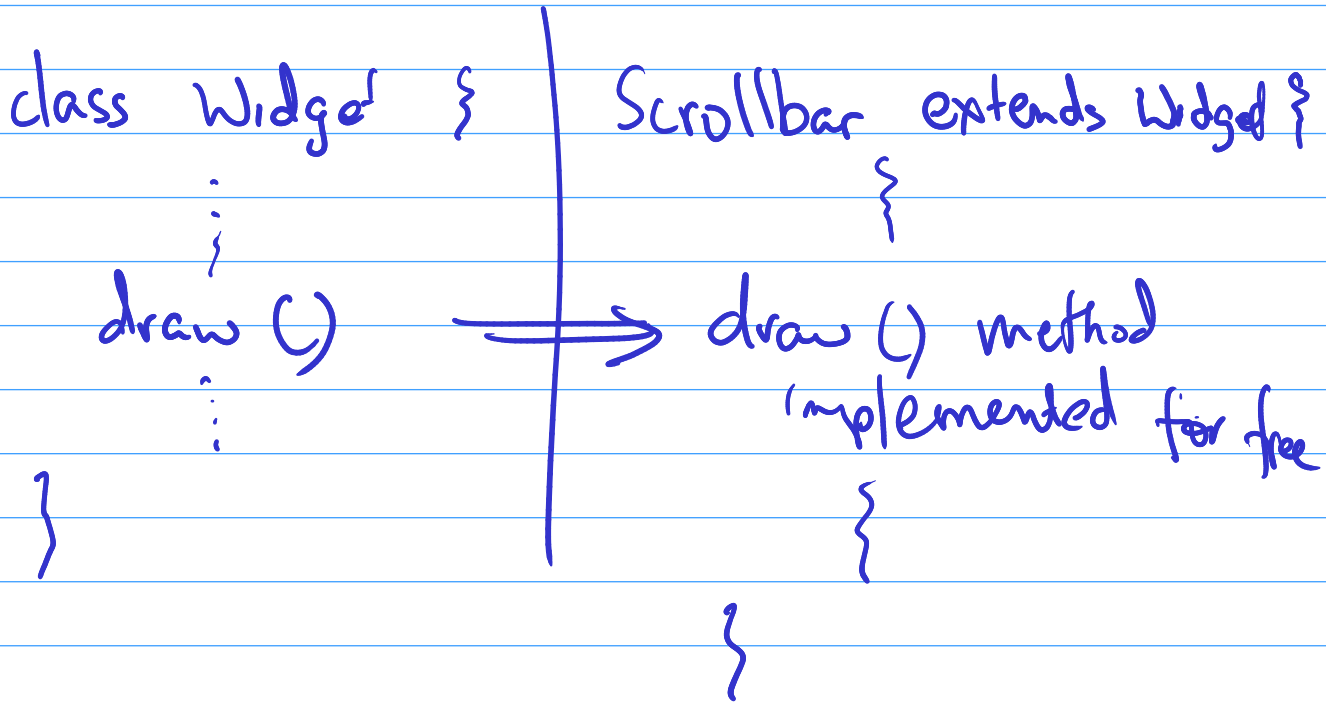
C++: namespaces scopes

Java: packages

Sense 3: Abstraction: Components interact through well-defined boundaries

Classes, Interfaces, Traits

Sense 4 : Reuse of code



Inheritance ("ad-hoc polymorphism")

Parametric polymorphism

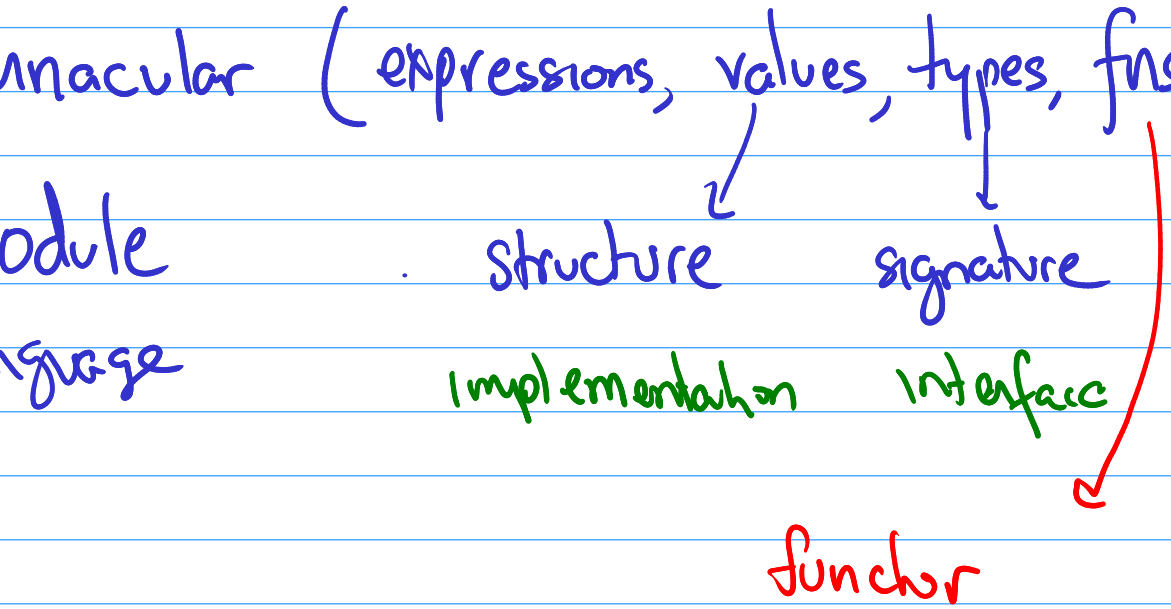
(OCaml, templates in C++,
generics in Java / C# / ...)

OCaml : Module System

Two sublanguages

① Vernacular (expressions, values, types, fns, ...)

② Module language



C : Core / Preprocessor

C++ : Core / Templates / Preprocessor

Lisp : Core / Macros