

```
fun append (xs,ys) =  
  if xs=[]  
  then ys  
  else (hd xs)::append(tl xs,ys)  
  
fun map (f,xs) =  
  case xs of  
    [] => []  
  | x::xs' => (f x)::(map(f,xs'))  
  
val a = map (increment, [4,8,12,16])  
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman
2013

Generics Versus Subtyping

What are generics good for?

Some good uses for parametric polymorphism:

- Types for functions that combine other functions:

```
fun compose (g,h) = fn x => g (h x)
(* compose : ('b -> 'c) * ('a -> 'b) -> ('a -> 'c) *)
```

- Types for functions that operate over generic collections

```
val length : 'a list -> int
val map : ('a -> 'b) -> 'a list -> 'b list
val swap : ('a * 'b) -> ('b * 'a)
```

- Many other idioms
- General point: When types can “be anything” but multiple things need to be “the same type”

Generics in Java

- Java generics a bit clumsier syntactically and semantically, but can express the same ideas
 - Without closures, often need to use (one-method) objects
 - See also earlier optional lecture on closures in Java/C
- Simple example without higher-order functions (optional):

```
class Pair<T1,T2> {  
    T1 x;  
    T2 y;  
    Pair(T1 _x, T2 _y) { x = _x; y = _y; }  
    Pair<T2,T1> swap() {  
        return new Pair<T2,T1>(y,x);  
    }  
    ...  
}
```

Subtyping is not good for this

- Using subtyping for containers is much more painful for clients
 - Have to **downcast** items retrieved from containers
 - Downcasting has run-time cost
 - Downcasting can fail: no static check that container holds the type of data you expect
 - (Only gets more painful with higher-order functions like **map**)

```
class LamPair {  
    Object x;  
    Object y;  
    LamPair(Object _x, Object _y) { x=_x; y=_y; }  
    LamPair swap() { return new  
        LamPair(y,x); }  
}
```

```
// error caught only at run-time:
```

```
String s = (String) (new LamPair("hi",4).y);
```

What is subtyping good for?

Some good uses for subtype polymorphism:

- Code that “needs a Foo” but fine to have “more than a Foo”
- Geometry on points works fine for colored points
- GUI widgets specialize the basic idea of “being on the screen” and “responding to user actions”

Awkward in ML

ML does not have subtyping, so this simply does not type-check:

```
(* {x:real, y:real} -> real *)  
fun distToOrigin ({x=x,y=y}) =  
    Math.sqrt(x*x + y*y)  
  
val five = distToOrigin {x=3.0,y=4.0,color="red"}
```

Cumbersome workaround: have caller pass in getter functions:

```
(* ('a -> real) * ('a -> real) * 'a -> real *)  
fun distToOrigin (getx, gety, v) =  
    Math.sqrt((getx v)*(getx v)  
              + (gety v)*(gety v))
```

- And clients still need different getters for points, color-points