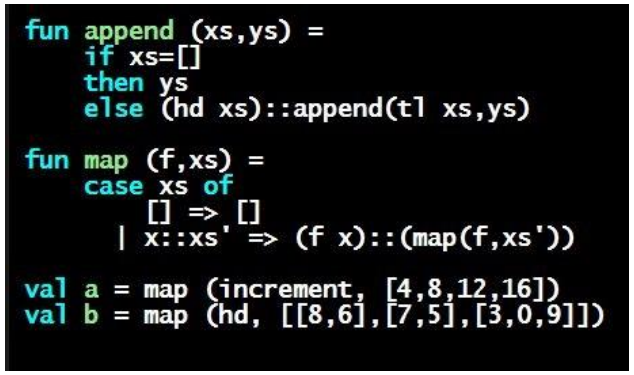

Programming Languages

Dan Grossman

A Little Type Inference

A new way to go

- For homework 2:
  - Do not use the `#` character
  - Do not need to write down any explicit types

- These are related
  - Type-checker can use patterns to figure out the types
  - With just `#foo` or `#1` it cannot determine "what other fields"

Why no problem

Easy for type-checker to determine function types:

```
fun sum_triple (x, y, z) =
    x + y + z

fun full_name {first=x, middle=y, last=z} =
    x ^ " " ^ y ^ " " ^ z
```

Get error message without explicit type annotation:

```
fun sum_triple (triple : int*int*int) =
     #1 triple + #2 triple + #3 triple

fun full_name (r : {first:string, middle:string,
                    last:string}) =
      #first r ^ " " ^ #middle r ^ " " ^ #last r
```

Unexpected polymorphism

- Sometimes type-checker is “smarter than you expect”
  - Types of some parts might be less constrained than you think
  - Example: If you do not use something it can have any type

```
(* int * 'a * int -> int *)
fun partial_sum (x, y, z) =
    x + z

(*{first:string, last:string, middle:'a} -> string*)
fun partial_name {first=x, middle=y, last=z} =
    x ^ " " ^ z
```

- This is okay!
  - A more general type than you need is always acceptable
  - Assuming your function is correct, of course
  - More precise definition of “more general type” next segment