

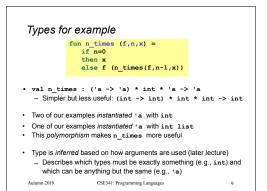
Relation to types

- Higher-order functions are often so "generic" and "reusable" that they have polymorphic types, i.e., types with type variables
- · But there are higher-order functions that are not polymorphic
- And there are non-higher-order (first-order) functions that are polymorphic

8

 Always a good idea to understand the type of a function, especially a higher-order function

Autumn 2019 CSE341: Programming Languages

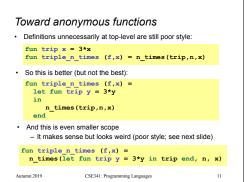


Polymorphism and higher-order functions . Many higher-order functions are polymorphic because they are so reusable that some types, "can be anything" . But some polymorphic functions are not higher-order _ Example: len : 'a list -> int . And some higher-order functions are not polymorphic _ Example: times_until_0 : (int -> int) * int -> int . fun times_until_zero (f, x) = if x=0 then 0 else 1 + times_until_zero (f, f x) . Note: Would be better with tall-recursion

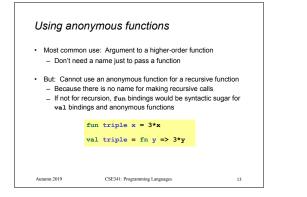
CSE341: Programming Languages

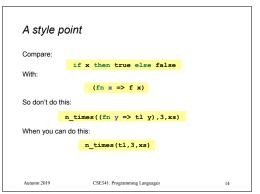
10

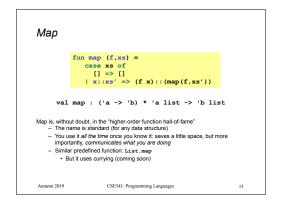
Autumn 2019

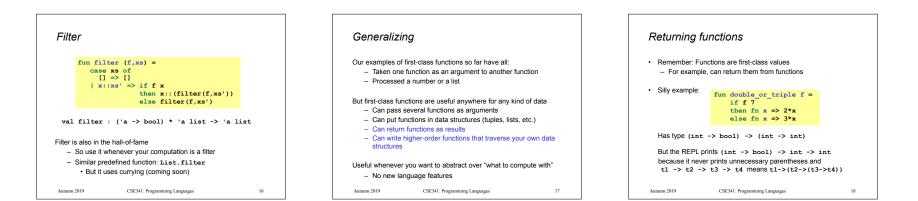


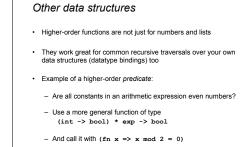
Anon	rmous functions	
This d	oes not work: A function binding is not an expressi	on
	<pre>triple_n_times (f,x) = times((fun trip y = 3*y), n, x)</pre>	
	s the best way we were building up to: an expressi conymous functions	on form
	<pre>triple_n_times (f,x) = times((fn y => 3*y), n, x)</pre>	
	ke all expression forms, can appear anywhere yntax: • fn not fun	
	• => not =	
	 no function name, just an argument pattern 	
Autumn 2019	CSE341: Programming Languages	12











CSE341: Programming Languages

Autumn 2019

19