Subtyping

$D \to R$ we want it to work on things "like" D but not exactly D

$nat \to nat$

$(nat \to nat) \to nat \quad \Leftarrow \quad \lambda f.\ 5 \quad : \forall \alpha.\ \alpha \Rightarrow num$

$\lambda f.\ f(6)$

most general polymorphic type (principal types)

type inference provably discovers ↑

$(nat \times nat) \to nat \quad \Leftarrow \quad \lambda p.\ (fst\ p) + (snd\ p)$

$\forall \alpha.\ (nat \times \alpha) \to nat \ : \quad \lambda p.\ 5 + (fst\ p)$

Cat = "Animal" × CatStuff

Animal = "Animal" × Nothing

"Duck Typing" ≈ "A type is what you can do to it"

~~in python~~ in Lua/Ruby, everything is a dictionary, capabilities = keys in dictionary

Subtyping so far =

Structural Subtyping

Objects have fields (methods are function-valued fields)

One object is compatible with another if shared fields are compatible

Obj A = {a, b, c, d, e, f}
Interface X = {a, b, c}
Y = {c, d, e}
Z = {d, e, f}

O ≤ X
Y
Z

OCaml, Haskell, Typed Racket, ie egghead academic languages

Java = Nominal Sub-typing

An object implements interfaces
(fields) (method sets)

One object O is compatible with interface I if O implmts I

O = {f...} × List Int
Int = {m...}

O = {a, b, c, d, e, f} × [X, Z]
O ≤ X O ≤ Z O ≰ Y