Functional Programming
W(hat)TF?
Lambda Calculus
\[ f(x, y) = x^y \]

\[ \lambda x. \lambda y. x^y \]
mathematical functions
Math:
\[ b = \{ n \mid n \in a \land n \leq 10 \} \]

Haskell:
\[ b = [n \mid n \leftarrow a, n \leq 10 ] \]
\[ f(x) = x^2 \]

Haskell: \[ f \ x = x^2 \]

Clojure: \( (\text{defn } f [x] \ (* x x)) \)
variables !

= variable

2x = 6 => x = 3
functions don't travel business class

first class citizens
partial application

```
multiply = -> x,y {x*y}
bytwo = multiply.curry[2]
bytwo[2] # => 4
```
compose

(like you are Mozart)
\[ f(g(x)) = (f \circ g)(x) \]

Haskell: \( f \cdot g = \lambda x \rightarrow f \,(g \ x) \)
smart people are lazy
smart languages too
Haskell:

\[
\text{fibs = 0 : 1 : zipWith (+) fibs (tail fibs)}
\]

\[
\text{take 10 fibs}
\]
\[
\Rightarrow [0,1,1,2,3,5,8,13,21,34]
\]

\[
\text{numbers = 1 : map (+1) numbers}
\]

\[
\text{take 10 numbers}
\]
\[
\Rightarrow [1,2,3,4,5,6,7,8,9,10]
\]

\[
\text{filter even (takeWhile (<40) numbers)}
\]
\[
\Rightarrow [2,4,6,8,10,...]
\]
recursion (n):
see recursion
fac :: Integer -> Integer
fac 0 = 1
fac n = n * fac (n - 1)
take :: Integer -> [a] -> [a]
take n _ | n <= 0 = []
take _ []   = []
take n (x:xs) = x : take (n-1) xs
W(why)TF?
referential transparency
W(ho)TF?
ML-like

Standard ML
OCaml
Haskell
statically typed

pattern matching

code is not data
Lisp-like

Common Lisp
Scheme
Clojure
dynamically typed

homoiconic (code == data)

lists
time to play!