f x = if x > 3 then 'a' else 'b'

f x

| x > 3 = 'a'

| otherwise = 'b'

printsums :: [[Int]] -> IO ()

printsums [] = putStrLn "Empty!"

printsums xss = mapM\_ printSum (zip [1..] xss)

where

printSum (i, xs) = putStrLn $ show i ++ ": " ++ show (sum xs)

largers :: Ord a => [a] -> [a] -> [a]

largers = zipWith max

An **expression** is any code that evaluates to a value. It produces (returns) a result, and this value can be used, assigned to a variable, or passed as an argument. For example, x + 1 or len("Hello") are expressions because they compute and yield a value

A **statement** is an instruction that performs an action, but does not itself return a value. Statements are used to control the flow of a program, perform assignments, or invoke functions, but they do not yield a result that can be used as a value. For example, print("Hello") or a variable declaration like int x; are statements—they instruct the computer to do something, but do not themselves produce a value

.a = b = 3 (in C)

Imperative programming: Instructions change memory locations or registers. Branching instructions alter the flow of control

procedural programming: Programs are composed of bodies of code (procedures) that manipulate individual

data elements or structures. Procedures encapsulate complexity.

Expression: a sequence of symbols that can be evaluated to produce a value

side effect: something that happens in addition to the computation of an expression's value. It must be "observable". (hallmark of imperative programing)

functional paradigm: writing functions that are like pure mathematical functions

Haskell is a lazy and pure functional programming language/statically typed, with a very elaborate type system/not object-oriented in any way