Functional Programming in 40 minutes (Russ Olsen):

* Look at OOP: we’ve been using for many years to success, though it can be a bit messy
* When working with an application over time, it will inevitably become complicated: requirements change, add / remove segments of code, changing dependencies, etc.
* Though OOP has been shown to work, there are many “Yes, BUT…” situations and definitions
* Perhaps we can take all the stuff that works and refactor it using a new paradigm?
* Functions in mathematics are relations between sets; they aren’t computed they just are. For every given input, there is always the same output. They don’t change, they are things (“objects”), unlike computer functions (computer procedures, methods, etc.).
* Can we use this idea of strictly defined / “pure” functions in our code?
* Apply these rules for functions: take input, produce output, no side effects!
* We do this in hopes to make our code easier to understand ☺
* Rule #1: Our data structures are immutable!!! There is no question of what state a value is in, it will always be the same. When we want to mutate a data structure, we make a copy of it with the changes you’d like to make.
* Persistent data structures – They have two characteristics: i. they are immutable, ii. Support an efficient version of the “copy on modification” operation, without much copying!
* This idea of persistent data structures is interesting, as they can be copied based upon the modification and the unused segments can be reused to form the new data structure (read this article potentially: <https://www.geeksforgeeks.org/persistent-data-structures/>)
* There is a problem: how do we get anything done? We don’t have side effects! We need a bridge from functional programming to the outside world
* Clojure has “atoms” as the closest thing to variables, which use functions to update state
* Clojure affects the outside world by again using functions to update outside states
* Functional programming: the programming you know with three unbreakable rules: functions, immutability and bridges to produce effects
* Functional programming isn’t magic! You still have problems: program errors, redundant code, bad code, etc. (maybe discuss how functional programming improves threads incredibly, with one thread being unable to wrongfully change the state of the program due to immutable data structures).
* The whole point is that functional programming limits these problems
* It can be hard to do at first, but the benefits generated and understandability of your code are very valuable

Functional Programming: Concepts, Advantages, Disadvantages and Applications:

* Functional programming is a programming paradigm in which it is tried to bind each and everything in pure mathematical functions.
* It is a declarative type of programming style that focuses on what to solve rather than how to solve
* Functional programming is inspired by lambda calculus, invented by Alonso Church.
* Lambda calculus is a formal system of mathematical logic for expressing computation based upon function abstraction and application using variable binding and substitution.
* Lambda calculus is Turing complete, meaning that anything you can do with a Turing machine (aka the classical computer) you can do with lambda calculus, and vice versa
* You can think of lambda calculus as a mathematical system for defining and applying nameless functions, referred to as anonymous functions (for example, the anonymous function λx.x defines the function f(x) = x , and λx.y defines the function f(x)=y, and λx.x+y defines the function f(x)=x+y).
* Instead of statements, lambda calculus makes use of expressions. Unlike a statement, which is executed to assign variables, the evaluation of an expression produces a value.
* Lambda calculus: computations with functions
* Functional programming concepts: pure functions, recursion, referential transparency, functions are first class and can be higher order, variables are immutable