What’s the big deal about functional programming?

Why functional programming?

- Let’s say you have developed a matrix manipulation framework that runs on a cluster of 1000 nodes.
- It can multiply matrices together, transpose them, invert them , transform their elements from one domain to another, all that stuff.
- You will need a driver which allows the programmer to set up the matrices and define operations such as multiplication, etc.
- All of those are easy to implement except *transform*. How exactly are you going to let your programmer specify what operation should be performed on an element of a matrix *when he doesn’t have direct access to an element of a matrix?*

Functional composition and higher-order functions

- A higher order function is a function, at least one of whose parameters is a function.
  - m2 <- m1 map f1
  - s <- m2 reduce f2

- Functional composition is where the result of a higher order function is itself a function.
  - f1 <- g andThen h

But how do we define the functions we need?

- Lambda calculus (lambdas for short).
- A lambda is an anonymous function which defines how its parameters are transformed into its result.
- You can find lambdas in the following languages…
    - Java8
    - Python
    - Scala
    - Haskell
    - Pharo
    - All functional programming languages

But that’s not all…

- There are many other aspects of functional programming, many of which we can’t find in Java8 or Python:
    - Lazy (deferrred) evaluation;
    - Type inference and shape preservation;
    - Pattern-matching;
    - Referential transparency:
        - Immutability by default;
        - Pure functions (lack of side-effects);
    - Tail recursion;
    - Tuples;
    - Monads, etc.
    - Higher-kinded types.

More later…