***Functional Programming:***

* Complexity is the thorn in every programmers side
* Functional programming is a style of programming where every function’s behavior is solely dependent on its input, completely independent of system state

***Imperative:*** Programs are created by a collection of ordered statements that control flow and compute values.

***Object-Oriented:*** Programs are organized into classes (each of which maintains state) and the methods that work on them.

***Functional:*** Programs are organized into functions, each of which produces output from input and cannot change system state.

* Imperative is procedural programming, it’s essentially just writing everything without functions
* Nouns are represented by variables, values can change according to the needs of the program.
* Verbs are represented as functions
* Object-Oriented Programming is an extension of imperative programming
* Object-Oriented Programming, classes should be nouns
* Object-Oriented Programming, Methods are the verbs.
* Functional Programming, Functional programming is a style of programming where applications are assembled from functions. Verbs are at the center
* Output is completely determined by input,
* Referential Transparency is achieved by when there are purely functions and the program is purely immutable
* Functions cannot be interactive coupling, this would mean that they are altering the state of the program
* The following disqualifies functions from being pure: Global Data, Files, Random Number Generators, User Interaction, By-Reference
* In the functional programming world, all variables must be immutable
* First-Order Function: Takes data and returns data
* High Order Function: A function that works on another function, or it returns a function
* A lambda is an anonymous function that doesn’t receive a name
* Lambdas can be passed as a parameter without needing a formal declaration.
* Filter: Creates a new collection from an old one based on criteria, filter expects two parameters, the function the old collection and the function that does the operating
* Applied function returns a bool
* Filters reduce the collection based on a criteria established in the function working on it.
* Map creates a new collection out of an old one expects the same parameters as a filter
* Maps are meant to transform data
* Reduce takes 2 parameters or more and returns one
* Reduce combines data
* Functional Programming is really good at helping reduce complexity in programming
* It helps minimize side effects and the impact on the state
* All functions in your design should be pure with few exceptions. Each exception should have a very good reason for breaking such a rule. Common examples include file I/O, user I/O, and random number generators. If a function is not pure and does not fall into one of these categories, then careful consideration should be taken as to why that is the case
* Recursion can reduce complexity and have minimal system impact when introduced appropriately
* Favor iteration over recursion
* Master Recursion for functional programming
* Pass functions as parameters
* Use Filter, Map, and Reduce