Functional Programing

JavaScript

**JavaScript Basics** (80% of code is written with 20% of JS)

* **Primitive data types in javascript**.
  + Numbers, String, Booleans and undefined
* **Complex Types** 
  + Object Literals, Dates, and Arrays
* **Objects** 
  + Objects{} are a record of data that are grouped together
* **Arrays** 
  + Arrays[] store other data types
* **Functions** 
  + Functions allow you to create ***reusable logic*** or code that ***transforms values*** for one thing to another like x => y
  + Functions can take in input parameters
  + Functions themselves are ***values*** just like how numbers, strings, booleans, and objects are values.
  + Functions are able to be passed into other functions
  + Passing functions into other functions are big part of ***functional programming***

**Immutable Data Structures** (Data that never Changes)

* **Is Immutable data good?** 
  + Immutable data is simpler because it never changes
  + Data that never changes result is less complicated code
  + Easier to understand, test, and maintain
  + Creating simple code isn’t easy but worth it in the long run
  + JS doesn’t support immutability well and doesn’t care
  + Create immutable code is on you.
  + Const variable only prevents reassignment
  + Const doesn’t make the data immutable
* **Updating objects, in an immutable way**
  + ***State*** are things your apps remembers
  + Things usually remembered are adding, updating, deleting in Apps
  + Keeping track of these kind of changes are called ***Maintaining State***
  + Updating Object in JS the immutable way use ***...Spread Operator***
  + Spread Operator is like copying an object inside of the new object
  + The spread operator allows you to ***add new property*** and ***update property*** without changing the original object
  + To ***delete properties*** in a object the immutable way use ***Destructing*** + ...***Rest*** operator
  + Destructingallows you to pull properties out of an object
  + The Rest Operator used inside of destructing allows you to pull properties out and return a object with the property removed
  + This allows you to remove properties in object without changing the object
  + When the (...) is used ***after*** the = is the Spread operator
  + When the (...) is used ***Before*** the = is the Rest operator
* **Updating Array, in an immutable way**
  + The ***spread*** Operator can as well be used on arrays [...] to ***add*** new properties to an array
  + The ***Map*** function is used to ***Update*** arrays
  + Arrays usually store more then one object inside of them
  + ***Map*** functions allows you to pass in a functions that updates each iterate and update the properties in the array and return a new array
  + The ***Filter*** function is used to ***Remove*** properties inside of an array
  + ***Filter*** takes in function that has argument which is going to determine what items to keep and what to remove
  + The **Reduce** Function Summarize the information inside of an array
  + ***Reduce*** Summarize the data in arrays
  + The function inside of ***reduce*** that compares the first parameter to the second then replace the first parameter with the return value
  + That first parameter is called the ***accumulator***
  + The ***accumulator*** can be anything you want, to give the ***accumulator*** a value you want it to use just pass that value in the second parameter in the ***Reduce***

**Types of Functions** (Functional programming style)

* **Currying** 
  + ***Currying*** transforming a function that takes multiply arguments to a function that takes a single argument
  + ***High-order Functions*** are functions that either take in functions in the parameter or returns a functions inside of it.
  + ***Closure*** are functions that can access and use variable that aren’t directly passed in the function because of the placement of the function relative to the variables
  + ***Partial Application*** specialize a general function, its used to pass data to curried functions
* **Partial Application** **&** **Currying** 
  + ***Currying*** is whats done to the function before you actually use that function, there is no datain the function
  + ***Partial Application*** is whats done to the curried function ,it passes in data
  + You can also use partial application on functions that aren’t curried functions
  + The order of ***Parameters*** matter in currying
  + Any ***Parameters*** that takes in a general function to a specialized function should be the first parameter
  + ***Ramda*** is library that simplifies currying functions
  + Ramda functions are curried by default and it never mutates data
* **Pure Functions & Impure Functions**
  + There are generally two kind of functions, Pure Functions and Impure Function which could also be called a ***Procedure***
  + Functional Programs almost always use pure functions
  + ***Pure Functions*** creates & returns values only based on inputs parameters and causes no side affects
  + **Pure Functions Rules**
    - * 1. Must have input Parameters
        2. No stateful values, the function shouldn’t have any variable outside of themself that depended on it to work because that variable could change over time
        3. Return should only be based on input
        4. No side affect, ***Side effect*** are code that causes change outside itself
  + Pure functions are not easy, but have a lot of benefit then writing ***Procedure***
  + **Pure Functions Benefits**
    - * 1. ***Reusable***, Pure functions can be reused
        2. ***Composable*** which mean you can combined functions to create new functions
        3. ***Easy to Test*** since you just provide input values and see if the result is what you expected
        4. ***Easy to Cache*** since you always get same result for the given input, its easy to cache expensive function calls
* **Function Composition**
  + Function composition is making new functions out of other functions
  + ***Ramda*** makes composition easier with its compose function.
  + The ***Compose*** function allows you to combine 2 or more pure functions and return a single function
  + The ***Compose*** function works right to left