Name Splitting (7m)

1. Explain parse name problem
2. Show imperative parse name
   * Side effecting against Name
   * String building middle name imperatively
   * Stating how to get the names we want, not what we want
3. Show functional parse name
   * No side effects
   * LINQ allows us to declare what we want rather than write how to get it
   * Ternary expressions allow us to do conditional logic as expressions
   * Aggregate combines all the names into a single string
   * Sloppy because it requires a trim in the case of single middle name
4. Show string.join version
   * Better, but hard to read
5. Show refactored version
   * Best. Use extension methods to keep the fluent style
   * Where C# starts to show lack of functional roots
   * Functional languages have a function composition operator to make this easier

State Change History (5m)

1. Explain state change problem
   * Show comment diagram for duration calculations
2. Show imperative version
   * Stating how to get the result
   * Lots of side effects.
   * Got to run the code in your head to keep track of that state
   * Lot of code!
3. Show functional version
   * Refer back to diagram. Think in sequences
   * Set up the two sequences
   * Use Zip to loop through both together
   * No side effects, cleaner
   * Stating literally what you intend, not how. Declarative coding

Maybe (4m)

1. Show null checking lines
   * Null checking typically requires statements
   * Side effecting – keep track of state
2. Show maybe extensions methods
   * Higher order function
   * One for reference types
   * One for nullable value types
3. Show expression based version
   * Maybe allows conditional access based on nullness
   * Null coalesce operator provides the fallback value
   * No side effects – totally expression based
4. C# 6 conditional access operator: ?.
   * Bringing more functional programming into C#
5. Maybe is still useful in C# 6
   * Conditional access is good only for member access
   * Show imperative example
   * Maybe lets you transform the value at each step

Name Splitting in JavaScript

1. Talk about lodash
   * JavaScript’s standard libraries are anaemic
   * 3rd party libraries to the rescue
   * Lodash provides many standard higher order functions
2. Review name splitting example
   * Functional concepts transfer cleanly to another language
   * Very similar code
   * First attempt at function composition in Lodash used “chain”
3. Introduce ES6 arrow functions
   * Another functional feature getting imported into JavaScript
4. Show v2 code of lodash using lodash/fp
   * v4 of Lodash includes FP-style API by default
   * Chain required full lodash import; no longer necessary
   * Use flow instead of chain
5. Explain how it works: function currying
   * myDrop example
6. Much more composable approach
   * You can write your own functions and compose them
   * Eg. Replace drop with myDrop

Duplicated Emails

1. Explain example – locate emails that occur more than once in the list of people
2. Can still import all of lodash if you want
3. Lodash provides functions that are idiomatic to JavaScript
   * Countby produces an object where the property names are the email addresses and the property values are the counts
   * PickBy creates a new object keeping only the properties that meet the predicate
   * Keys returns an array of all keys on an object
4. FP benefits
   * Stating what you want – declarative programming
   * Code is clearer
   * Yes, there’s a burden in knowing what the functions are
   * However, it’s common language between projects and programmers
   * Lift the level of abstraction – more productivity and more code reuse

Rx – User-driven Search

1. Explain difference between LINQ and Rx
   * LINQ and lodash are functional libraries for dealing with sequences of items
   * Rx is a functional library for dealing with sequences of events
2. Explain example – user driven search.
   * Each change to the textbox causes a search to occur
   * Bad example - server responses out of order
   * Demonstrate (http://localhost:3000)
3. Show the code (userSearchBadCtrl.js)
   * Search gets called for each change in the textbox
   * Fires a web request that updates asynchronously
4. Demo fixed RxJS version
5. Show the code (userSearchGood.js)
   * Using rx.angular to add createObservableFunction
   * Creates a function that fires an event every time it is called
   * Returns the observable sequence
   * Flatmaplatest: like selectmany – flattens a sequence of sequences. The promise is the inner sequence
   * “Latest” means only pass on the events from the last inner sequence
6. Insert debounce(200) to clean up number of requests

Rx - Server Polling

1. Explain example
   * Poll the server periodically to see if an operation has completed
   * State change will probably happen soon, so poll quickly, then slower if the user leaves the page open
   * Stop polling once completed
2. Easy to do when thinking of sequences of events
3. Demo
4. Show code – serverPollingCtrl.js
   * Create observable sequence by concating sequences together
   * Subscribe to it, use flatMapLatest and query server
   * If state is completed, dispose subscription – stops polling
5. Functional programming advantages
   * composability and reuse
   * programming at a higher level of abstraction – operations over a sequence of events
   * declarative programming, what do we want rather than how to get what we want
6. Rx is available for many platforms, C#, Java, Scala, and others