

Functional Programming for your UI5 Apps Hands-On Worksheet

Any questions relating to this document should be addressed to:

DJ Adams
qmacro@gmail.com

Author	DJ Adams
Version	1.1
Date of Issue	21-06-2018

Table of Contents

[Table of Contents](#)

[Document Control](#)

[Introduction](#)

[Part 1 - Warming up](#)

[Part 2 - Currying and point-free code](#)

[Initial improvements](#)

[Currying and going point-free](#)

[Part 3 - Building Confidence](#)

[Final Thoughts](#)

[References](#)

Document Control

Version	Description	Editor	Date
0.1	Initial draft	DJ Adams	24-06-2017
1.0	First release version	DJ Adams	26-06-2017
1.1	Minor modifications for 2018, including a new end in the Ramda REPL	DJ Adams	21-06-2018

The canonical link to this worksheet is

<https://docs.google.com/document/d/1Nx2PFqObMtirOrSzjU804PAAVkc3j4lZTtfRRoLSocQ/edit#>.

It is also available at the short link <http://bit.ly/qmacro-ui5con-funcprog>.

Introduction

This document is a worksheet to accompany the hands-on session "[Functional Programming for your UI5 Apps](#)" at [UI5con](#) on 29 June 2017, in SAP building ROT03, St Leon Rot and to be repeated on 22 June 2018 at the same event and in the same place a year later. You can access this document via the short link <http://bit.ly/qmacro-ui5con-funcprog>.

The code is based on various versions of OpenUI5 and the versions are indicated where appropriate. (The reason for this is minor - as the SDK changes over time, the accuracy and relevance of references may drift, but if we lock those references to the versions of OpenUI5 that they were based on, the relevance will remain.)

Code that you have to type in is **highlighted like this**.

Part 1 - Warming up

This part is based on [OpenUI5 version 1.46.9](https://openui5.hana.ondemand.com/1.46.9).

This is a brief introduction to thinking in terms of arrays and functions that operate on those arrays. Think "everything is a list". Think "pipeline". Think "immutability". Think "what, not how".

In this warm up, we'll grab some familiar data, and work on it inside the console of the Chrome Developer Tools. As we work on it, we'll execute code, and then examine and discuss it, and find ways to improve it. At the end we should be comfortable with processing arrays, with the functions filter and reduce (and by relation, map), and the concepts of higher-order functions and closures.

Step	Action
1	Open the 1.46.9 version of the UI5 Explored app (https://openui5.hana.ondemand.com/1.46.9/explored.html) and bring up the Chrome Developer tools.
2	Grab the entity data from the model supplying the UI <pre>aEntities = sap.ui.getCore().byId("__xmlview0").getModel("entity").getData(). entities</pre>
3	How many entities are there? Understand the relationship between entities, samples and namespaces by exploring the data, both in the console and in the Explored app. (There should be 176 entities, some with only 1 sample, others with more samples. Each entity belongs to one of around 20 namespaces.)
4	Calculate how many samples are there for entities in the sap.ui.core namespace. <pre>var total = 0; for (var i = 0; i < aEntities.length; i++) { var mEntity = aEntities[i]; if (mEntity.namespace === "sap.ui.core") { total = total + mEntity.sampleCount; } }</pre>

5	Discussion: What issues are there with this code, and with this approach?
6	<p>To prepare to build an alternative computation, start by filtering the list of entities to just those in the sap.ui.core namespace:</p> <pre>aEntities .filter(x => x.namespace === "sap.ui.core")</pre> <p>This should produce a shorter list of around 8 entities.</p>
7	<p>Now, with the focused list of sap.ui.core entities, count the total number of samples (and embrace the beauty of whitespace¹):</p> <pre>aEntities .filter(x => x.namespace === "sap.ui.core") .reduce((a, x) => a + x.sampleCount, 0)</pre> <p>There should be around 21 samples in total.</p>
8	Discussion: How does reduce differ from filter? Can we make this more generic?
9	<p>Create a helper function that will produce a function that can be used as a filter predicate.</p> <pre>by = (p, v) => x => x[p] == v;²</pre>
10	Discussion: What *is* that?
11	<p>Use it:</p> <pre>aEntities .filter(by("namespace", "sap.ui.core")) .reduce((a, x) => a + x.sampleCount, 0)</pre>
12	Discussion: Is by() the predicate function for filter?
13	Create a helper function to use with reduce:

¹ to create a newline in the Chrome Developer Tools console, use Shift-Enter

² for more information, please reread

	<pre>sumOf = p => (a, x) => a + x[p]</pre>
14	<p>Use it:</p> <pre>aEntities .filter(by('namespace', 'sap.ui.core')) .reduce(sumOf('sampleCount'), 0)</pre>
15	<p>Bonus: The reduce function is often used to turn an array into a scalar value, like we've used reduce thus far. But it doesn't have to. Write an expression to compute the number of entities in each namespace, ending up with a map:</p> <pre>aEntities.reduce((a, x) => { a[x.namespace] = (a[x.namespace] 0) + 1; return a; }, {})</pre>

Part 2 - Currying and point-free code

This part is based on [OpenUI5 version 1.42.9](#).


Now we're warmed up, we'll put the new approaches to computation to good use, in a sample UI5 app - the [Shopping Cart demo app](#) in the 1.42.9 SDK. In this part, we'll dig into the Shopping Cart app, and modify code in the controller on the fly, in the Chrome Developer Tools. In addition, we'll become familiar with currying, and point-free programming - that is, building and then using clean expressions that are free of data references.

The Shopping Cart has a master-master-detail app design; the first master shows a list of product categories such as Accessories or Graphics Cards, and the second master shows a list of products within the chosen category.

This part is in two sections. In the first section we'll identify a place for improvement, and make an initial change moving from "how" to "what". In the second section we'll bring in a curry function from Ramda, a functional programming library, and make it even better.

Initial improvements

Here we'll replace some procedural, imperative code with something more functional and succinct.

Step	Action
1	Open the Shopping Cart demo app (https://openui5.hana.ondemand.com/1.42.9/test-resources/sap/m/demokit/cart/index.html) and bring up the Chrome Developer Tools.
2	<p>If you have the Chrome Developer Tools docked on the right, it's likely you'll not have enough space left for the master to show in the app (you also probably had this problem in Part 1). Fix this by changing the mode of the Split App control that's hosting the master:</p>  <p>Use the Inspector picker and select the Split App's hamburger menu element, or something near it. In the Elements pane navigate up the hierarchy until you see the element that represents the Split App control; it will have an ID that ends "--splitApp", something like this:</p>

	<p><code>__xmlview0--splitApp</code></p> <p>Make sure it's selected (the element will be highlighted), and then hit Esc to toggle the console drawer, if it's not open already. Then get hold of the Split App control:</p> <pre>sa = sap.ui.getCore().byId(\$0.id)</pre> <p>Now set the mode of the Split App to "StretchCompressMode":</p> <pre>sa.setMode(sap.m.SplitAppMode.StretchCompressMode)</pre> <p>The master should now appear on the left, and not disappear, regardless of the available port width.</p>
3	This box left intentionally (almost) blank :-)
4	<p>Navigate to a product - pick a category and then a product within it. This will cause the Product view and controller to be loaded. Check in the Network pane of the Chrome Developer Tools to make sure.</p> <p>Open up the Product.controller.js using the Ctrl-P (Cmd-P on macOS) shortcut, or via the Navigator side panel in the Sources pane.</p>
5	<p>The <code>_addProduct</code> function is called when the "Add to Cart" button is pressed. Set a breakpoint (marked ">>" below) at the start of the first for loop here:</p> <pre>var oEntry = null; >> for (var i = 0 ; i < aCartEntries.length ; i ++) { if (aCartEntries[i].ProductId === oProduct.ProductId) { oEntry = aCartEntries[i]; break; } }</pre> <p>Hit the "Add to Cart" button and have a quick look what's happening.</p>
6	Discussion: What is this code doing?
7	Rearrange the if-the-else code block to allow things to run a little neater after we make changes. Change it from this:

```

if (oEntry === null) {
    // create new entry
    oEntry = {
        Id : jQuery.sap.uid(),
        Quantity : 1,
        Name : oProduct.Name,
        ProductId : oProduct.ProductId,
        ProductName : oProduct.Name,
        Price : oProduct.Price,
        SupplierName : oProduct.SupplierName,
        Status : oProduct.status,
        Weight : oProduct.Weight,
        PictureUrl : oProduct.PictureUrl
    };
    oCartData.entries[oCartData.entries.length] = oEntry;
} else {
    // update existing entry
    oEntry.Quantity += 1;
}

```

to this, so that the condition results are written the other way around:

```

if (oEntry) {
    // update existing entry
    oEntry.Quantity += 1;
} else {
    // create new entry
    oEntry = {
        Id : jQuery.sap.uid(),
        Quantity : 1,
        Name : oProduct.Name,
        ProductId : oProduct.ProductId,
        ProductName : oProduct.Name,
        Price : oProduct.Price,
        SupplierName : oProduct.SupplierName,
        Status : oProduct.status,
        Weight : oProduct.Weight,
        PictureUrl : oProduct.PictureUrl
    };
}

```

	<pre> }; oCartData.entries[oCartData.entries.length] = oEntry; } </pre> <p>Remember to save the code! Use Ctrl-S (Cmd-S on macOS) in the Chrome Developer Tools. Remember also that these modifications are ephemeral - if you reload the page the original resources will be fetched from the server again, and your changes will be lost ... so don't!</p>
8	Add a few more products to the basket to check that your changes have been made correctly and nothing's broken. Good.
9	Discussion: What stands out as "mechanical" in this function? Can we see any "how" that could be replaced with "what"?
10	<p>Bring about a more succinct "what" style to the part you set a breakpoint on earlier, using the <code>find</code>³ function, related to the filter, map and reduce functions. Looking at the existing code, what is its intention? To attempt to find an existing entry for the currently selected product⁴.</p> <p>Rewrite that in a more "what" style by replacing these lines:</p> <pre> var oEntry = null; for (var i = 0 ; i < aCartEntries.length ; i ++) { if (aCartEntries[i].ProductId === oProduct.ProductId) { oEntry = aCartEntries[i]; break; } } </pre> <p>with:</p> <pre> let oEntry = aCartEntries.find(x => x.ProductId === oProduct.ProductId); </pre>

³ beware - find is not supported in Internet Explorer. Then again, if you're using Internet Explorer (or even any part of MS-Office), shame on you - please leave the room.

⁴ there's even a comment line in there telling us that. Gasp!

11	Add more products to the cart, again, to check you haven't broken anything.
----	---

So far so good. At this stage we've made a nice improvement, but we can do better.

Currying and going point-free

The improvement with the find function is great, but with functional programming techniques we can make things even more succinct. This isn't about [Code Golf](#) - far from it. It's about making code easier to read and write, by making it shorter - less mechanical, and more declarative.

Using the [currying](#) technique we can go further on this journey, using [partial application](#) with [point-free style](#) code. There are functional programming libraries for JavaScript. My favourite is [Ramda](#) and we'll import it into our running Shopping Cart application in order to be able to use its [curry](#) function.

We could write our own curry function directly, but that would distract us and you'll probably want to use Ramda for other things anyway. Pick your level of abstraction and be proud of it!

To import Ramda, we'll just add a `<script>` element to the Document Object Model's `<head>`.

Step	Action
1	Create a <code><script></code> element pointing to the Ramda CDN: <code>1</code>
2	Add that <code><script></code> element to the document's head: <code>document.getElementsByTagName("head")[0].appendChild(ramdascript)</code> you should see the effect of this in the Network pane of the Chrome Developer Tools - the loading of the resource ramda.min.js from that CDN location. At this point, you can access Ramda and its functions with the upper case R variable.
3	Consider what is happening in the predicate function used in the call to find: <pre>let oEntry = aCartEntries.find(x => x.ProductId === oProduct.ProductId // <----);</pre>

	<p>We are saying: "For a given thing 'x', return a boolean value depending on whether the value of the specific property 'ProductId' for that thing 'x' matches the value of the same property 'ProductId' of a (reference) thing 'oProduct'".</p> <p>That sounds like a specific instance of something that could be more generic and thereby also more useful.</p>
4	<p>Let's make that statement more generic and express it in code:</p> <pre>(prop, ref, x) => x[prop] === ref[prop]</pre> <p>This is a definition of a function that takes three arguments: the name of a property ("prop"), a reference thing ("ref") and a thing-under-test ("x") ... and returns a boolean depending on whether the reference thing's property is the same as the thing-under-test's property.</p>
5	<p>Create a function that is a curried version of the above function statement, and add it after the variable declarations in the <code>_addProduct</code> function:</p> <pre>let hasSame = R.curry((prop, ref, x) => x[prop] === ref[prop])</pre>
6	<p>Use this new <code>hasSame</code> function to create another function that's specific to our circumstance, but still free of any specification of what things are to be under scrutiny:</p> <pre>let hasSame = R.curry((prop, ref, x) => ref[prop] === x[prop]), isSameProd = hasSame("ProductId", oProduct);</pre>
7	<p>Discussion: What's happening here? Can we visualise, or imagine, what functions are doing what, and expecting what arguments, and when?</p>
8	<p>Put our <code>isSameProd</code> function to use, by replacing this:</p> <pre>let oEntry = aCartEntries.find(x => x.ProductId === oProduct.ProductId);</pre> <p>with this:</p>

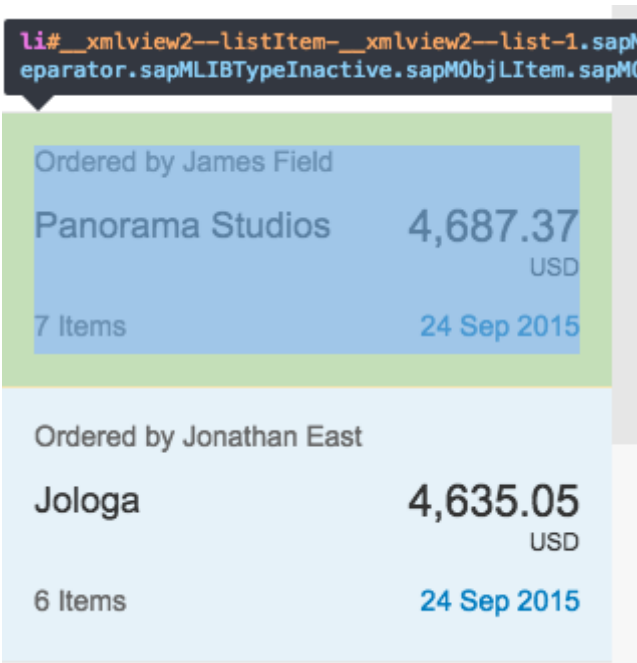
	<code>let oEntry = aCartEntries.find(isSameProd);</code>
9	Sit back and stare at this for a bit.
10	Don't forget to manipulate the shopping cart by adding (and perhaps removing) products, to ensure you haven't broken anything.

Part 3 - Building Confidence

This part is based on one of the sample applications in the SAP Web IDE - namely the Approve Purchase Orders app. This part doesn't introduce any new concepts, but allows us to practice some of the techniques we've learned in parts 1 and 2.

There's a part in the app's List Selector controller that could do with a little attention, to make it simpler. Again, it's a small example, but serves to illustrate the thinking we might benefit from in the wider context of a complete app.

Step	Action
1	Open the SAP Web IDE in your SAP Cloud Platform Account - a Trial Account will do. You do have a Trial Account, don't you?!
2	Create an instance of the Approve Purchase Orders app, with menu path File -> New -> Project from Sample Application Choose the "Approve Purchase Orders" selection and follow the wizard to completion. You should end up with a project called: sample.ApprovePurchaseOrders
3	Hit the Run button, and from the "Choose the File to Run" dialog, choose this file: flpSandboxMockServer.html Select the Approve Purchase Orders tile from the Fiori Launchpad to open the app.
4	Open the Chrome Developer Tools, and make the same adjustment to the Split App control as we did in Part 2 ... this time, the ID of the Split App control that we're looking for will look something like this: __xmlview0--approvalApp
5	Open up the List Selector source in the Chrome Developer Tools - use Ctrl-P (Cmd-P on macOS) to find the controller file ListSelector.js, and then find the function prepareResetOfList.

6	<p>Discussion: What do you think this function does? Is it easy to follow?</p> <p>The computation in the prepareResetOfList function deals with Purchase Order IDs (POIDs) - document numbers. But we can't see them in the actual app, so let's fix that by adding the POID to each of the items in the master list.</p>
7	<p>First Use the Inspector picker to choose an item in the list. Make sure you select the entire item, like this:</p>  <p>The ID should look like this:</p> <pre>__xmlview2--listItem-__xmlview2--poList-1</pre> <p>In other words, we've selected the second (list-1) (remembering it's a 0-based index) item.</p> <p>Check you've got the right element by inspecting the metadata of the selected control - it should be an Object List Item:</p> <pre>sap.ui.getCore().byId(\$0.id).getMetadata()</pre> <pre>> E {_sClassName: "sap.m.ObjectListItem", ... }</pre>

8	<p>Now get the item's parent, which should be the List control itself, and take from that the items aggregation binding information, which we'll store in variable b:</p> <pre>b = sap.ui.getCore().byId(\$0.id).getParent().getBindingInfo("items") > Object {path: "/PurchaseOrders", ..., template: f, ... }</pre>								
9	<p>Now we have the binding info we can hack the aggregation binding template to add a new Object Attribute to show the POId:</p> <pre>b.template.addAttribute(new sap.m.ObjectAttribute({text: "{POId}"))) > f {bAllowTextSelection: true, ..., sId: "__xmlview2--listItem", ... }</pre>								
10	<p>In order to see the POId in the master list, we'll have to cause a rebinding, which we can simply do by performing a search in the UI. Search for something, and then reset the search. When the items are re-rendered, we will see the POIds:</p> <div data-bbox="264 1028 863 1346" style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <p>Ordered by EPM User</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 60%;">C.R.T.U.</td> <td style="text-align: right;">6,615.21</td> </tr> <tr> <td></td> <td style="text-align: right;">USD</td> </tr> <tr> <td>9 Items</td> <td style="text-align: right; color: blue;">24 Sep 2015</td> </tr> <tr> <td>300000037</td> <td></td> </tr> </table> </div>	C.R.T.U.	6,615.21		USD	9 Items	24 Sep 2015	300000037	
C.R.T.U.	6,615.21								
	USD								
9 Items	24 Sep 2015								
300000037									
11	<p>Now we have the POIds visible, our examination of the prepareResetOfList function will make a bit more sense.</p> <p>Add the following variables to the "Watch" list, from the Sources pane in Chrome Developer Tools: aTail, aPreferredIds and sCurrentPOId ...</p> <p>... and set a couple of breakpoints like this:</p>								

```

105 // Prepare for the removal of some items from the list (due to approvals/rejections).
106 // This is done by setting the IDs currently in the list to preferredIds. Thereby we
107 // start with the item currently displayed. Then the IDs following this element are a
108 // in their current order. Finally, we add those items listed in front of the current
109 // order.
110 prepareResetOfList: function(oGlobalModel) {
111     var aListItems = this._oList.getItems(),
112         bFound = false,
113         aTail = [],
114         aPreferredIds = [],
115         sCurrentPOId = oGlobalModel.getProperty("/currentPOId");
116     for (var i = 0; i < aListItems.length; i++) {
117         var oItem = aListItems[i],
118             oCtx = oItem.getBindingContext(),
119             sPOId = oCtx.getProperty("POId");
120         bFound = bFound || sPOId === sCurrentPOId;
121         (bFound ? aPreferredIds : aTail).push(sPOId);
122     }
123     aTail.reverse();
124     aPreferredIds = aPreferredIds.concat(aTail);
125     oGlobalModel.setProperty("/preferredIds", aPreferredIds);
126     oGlobalModel.setProperty("/currentPOId", null); // Reset the current ID (we only
127 }

```

Cause the prepareResetOfList function to be invoked, by choosing an item in the master list and selecting the Approve button.

12 Discussion: What is happening in this function? What's the purpose of the for loop? How can we make this more succinct?

13 We can improve the function by embracing the fact that we're dealing with a list of items (remember, "everything is a list!") and swapping out the for loop for something more direct.

Make the following adjustments:

```

prepareResetOfList: function(oGlobalModel) {
    var aListItems = this._oList.getItems(),
        bFound = false,
        aTail = [],
        aPreferredIds = [],
        sCurrentPOId =
oGlobalModel.getProperty("/currentPOId");
    aPreferredIds = aListItems.map(x =>
x.getBindingContext().getProperty("POId"));
    for (var i = 0; i < aListItems.length; i++) {
        var oItem = aListItems[i],
            oCtx = oItem.getBindingContext(),
            sPOId = oCtx.getProperty("POId");
        bFound = bFound || sPOId === sCurrentPOId;
        (bFound ? aPreferredIds : aTail).push(sPOId);
    }
}

```

	<pre> aTail = aPreferredIds.splice(0, aPreferredIds.indexOf(sCurrentPOId)); aTail.reverse(); aPreferredIds = aPreferredIds.concat(aTail); oGlobalModel.setProperty("/preferredIds", aPreferredIds); oGlobalModel.setProperty("/currentPOId", null); // Reset the current ID (we only have preferences now) } </pre>
14	Discussion: Can we improve this further? Also, what is splice doing, and why doesn't it smell nice?
15	<p>Let's explore a less destructive, non-mutating way to split. Typically, the array <code>aPreferredIds</code> will contain a list of PO IDs, like this:</p> <pre> ["300000037", "300000038", "300000039", "300000036", "300000035", "300000033", "300000034", "300000032", "300000031", "300000029", "300000030", "300000028", "300000027", "300000026", "300000025", "300000024", "300000023", "300000021", "300000022", "300000020"] </pre> <p>In many functionally oriented languages there will be language elements directly for this. In Ramda, which we've seen briefly already, there's the splitAt function.</p>
16	Look at the documentation for <code>splitAt</code> : https://ramdajs.com/docs/#splitAt and determine in what ways it's different to plain JavaScript's <code>splice</code> .
17	<p>Try it out. Open the Ramda REPL with this link and you'll have already a couple of lines that looks like this:</p> <pre> const aPreferredIds = ["300000037", "300000038", ...] const sCurrentPOId = "300000039" </pre>
18	<p>Add the following line, to create two new constants that receive the left hand and the right hand side of the split list:</p> <pre> const [aLeft, aRight] = splitAt(indexOf(sCurrentPOId, aPreferredIds), aPreferredIds) </pre>

19	Examine what ends up in aLeft and aRight. Note also the different way round Ramda has the arguments to many of its functions. Why is that? That's for you to ponder!
----	--

Final Thoughts

The examples in this worksheet are deliberately simple, and have addressed, by and large, common patterns such as for loops. There is of course much more that can be gained by thinking functionally when writing or maintaining code in your UI5 apps.

We set out our stalls at the start with:

Think "everything is a list". Think "pipeline". Think "immutability". Think "what, not how".

Lists are incredibly simple, and through that simple structure and the functions that are available to operate on them, they are also incredibly powerful.

We saw a glimpse of what pipelines might look like in Part 1, when we passed the output of a call to filter to the input of a call to reduce. We didn't think or have to care about intermediate data structures, nor about cleaning up after ourselves. We didn't even need to think about the mechanics of processing items in that pipeline - the [list machinery](#) took care of that for us and allowed us to elevate our computational thinking to a higher level of abstraction.

We also saw implicit immutability, and explicit mutation in the last example in Part 3, which didn't feel right (and was put there deliberately to invoke that feeling!).

Hopefully, above all, we have felt what it's like to think in terms of "what", not "how".

References

For further reading, see these documents:

[Programming in a more functional style in JavaScript - Tech Workshop Notes](#)

[DEV219: Building More Stable Business Apps with Functional Techniques in JavaScript](#)