Adding Feedback to Concerto tests (v5.0.13)

Adding feedback to a test normally involves the following steps:

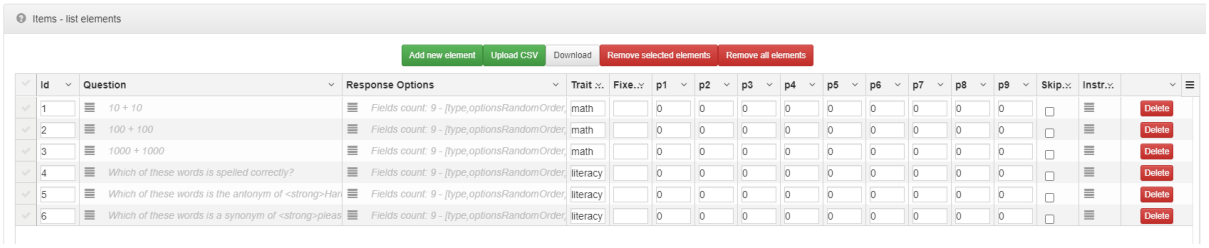
1. Obtaining values from the assessment node (e.g. raw score, theta)

2. Transforming those values in some way (e.g. standardising against a norm)

3. Visualising the new values in some way

This tutorial will show you a few different ways you can perform each of these steps, and some tips for code efficiency in cases where you want to show the same type of chart multiple times with different values, for example if you have a test with multiple traits.

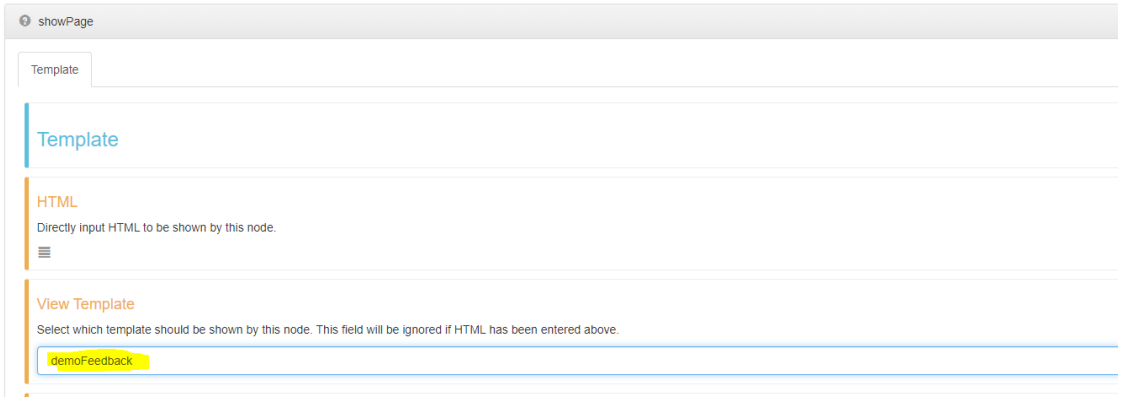
- I've made a simple linear test that measures 2 traits, with 3 questions per trait. Each question has only one correct answer, and I've defined this in the 'scoremap' for each question.



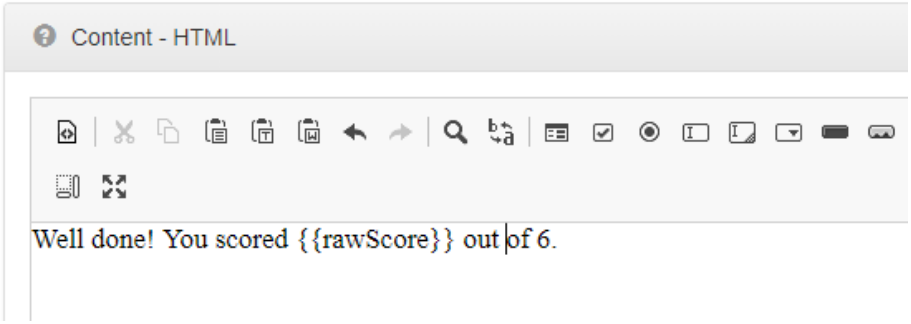
- I want to add a feedback page at the end that tells people their raw score out of 6. To do this I will:

- Create the template I'm going to use to show the feedback (Templates tab -> Starter content -> Edit 'page' template -> Copy -> Call it 'demoFeedback' -> Save.)

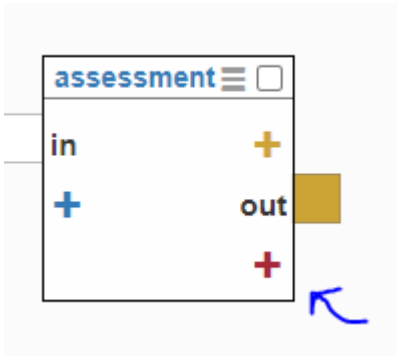
- add a showPage node to my test flow and set it to use 'demoFeedback' as the View Template

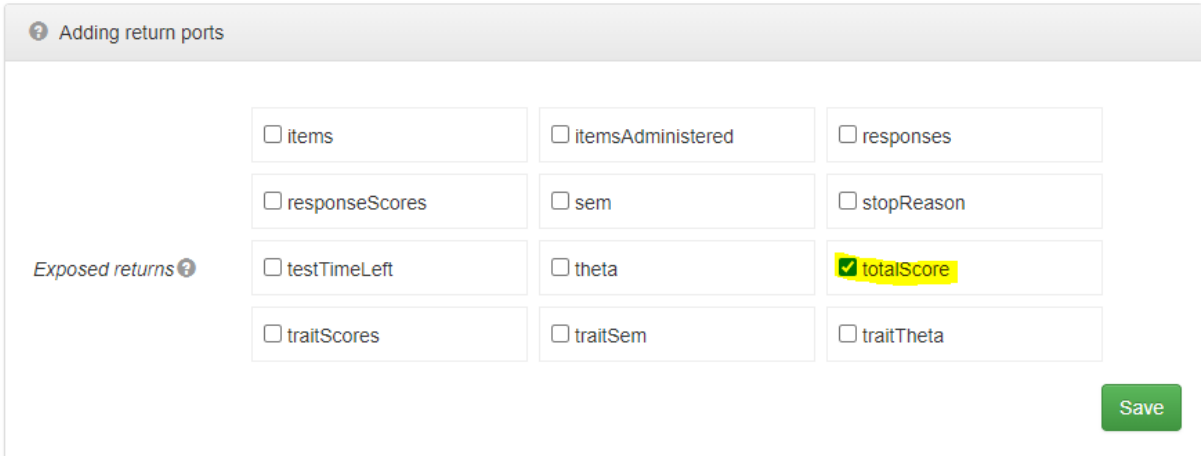


- In the 'content' for this View Template I will write the feedback I want to show, and insert the place where I want the raw score to go using curly brackets. Click Save.

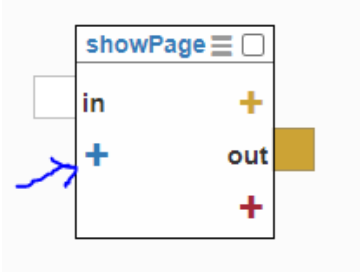


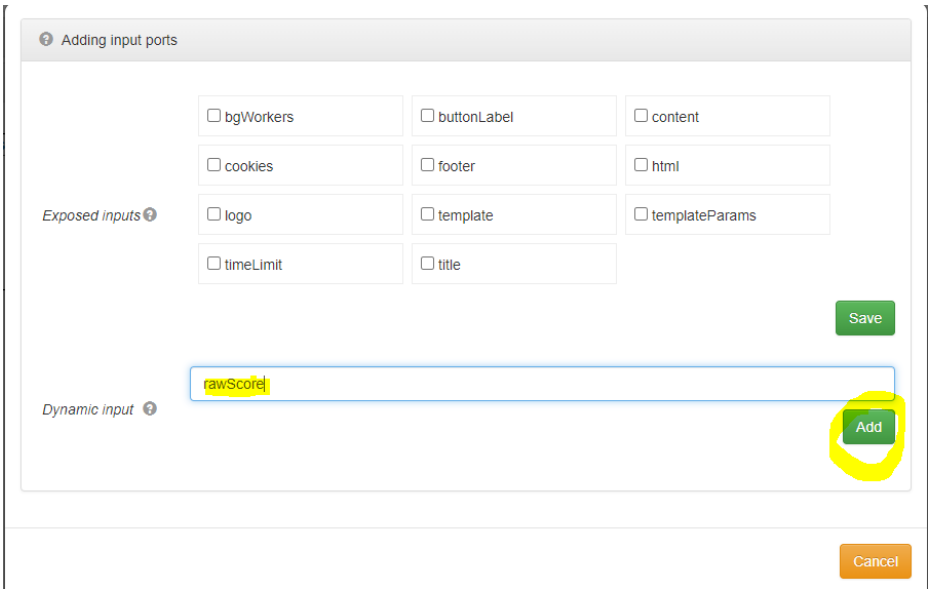
- Assessment node can already output raw score. Expose the return port called ‘totalScore’ and Save



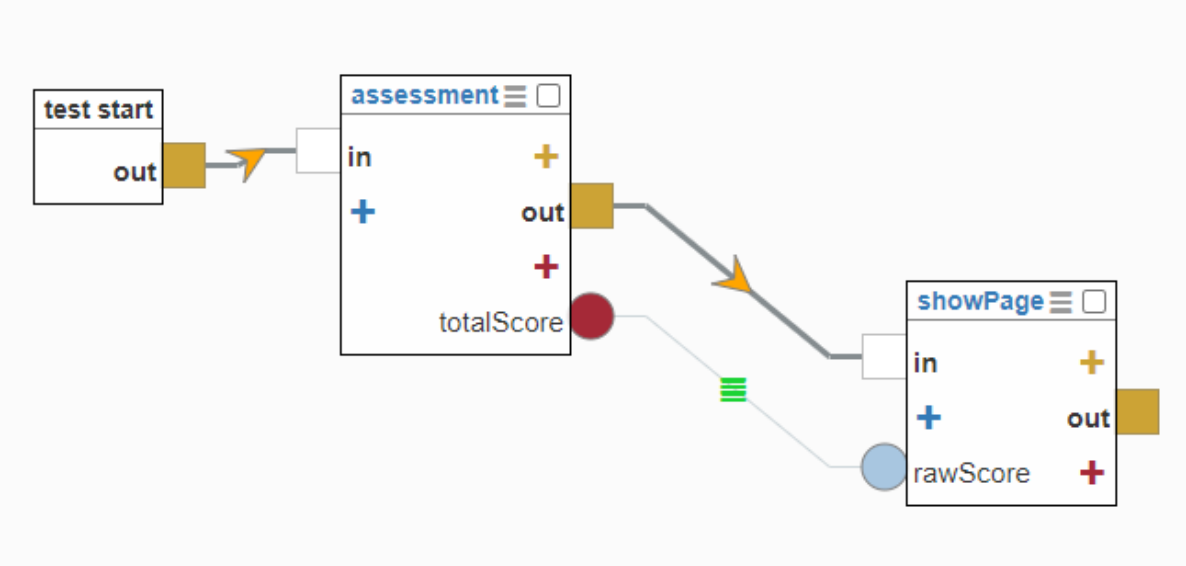


- We want to pass this totalScore variable from the assessment node to our new showPage node, so create a dynamic input port on your showPage node and call it rawScore (as this was the variable name we used in the curly brackets). In your own tests you can call this input port whatever you want, as long as it matches how you refer to it in the feedback template).

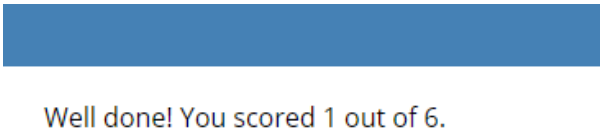




- Drag the totalScore output port from assessment node to the rawScore input port of showPage node. Connect the execution flow. Save. Your flow should look like this:



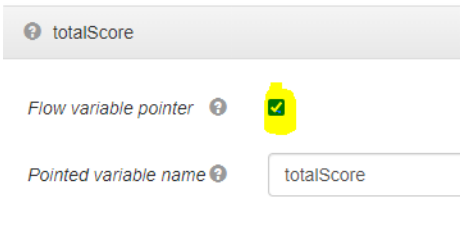
- When you run the test you should now see that the totalScore (i.e. the number of correct responses) is being filled into your feedback template content in the place where you put the curly brackets, like so:

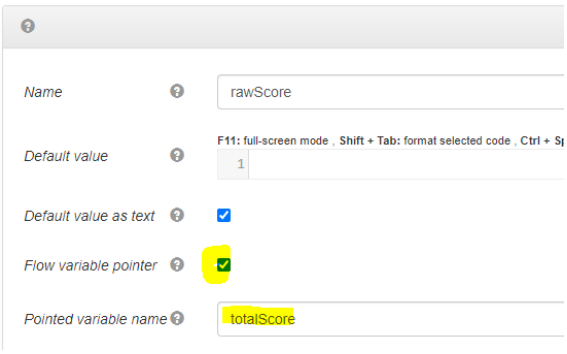


Well done! You scored 1 out of 6.

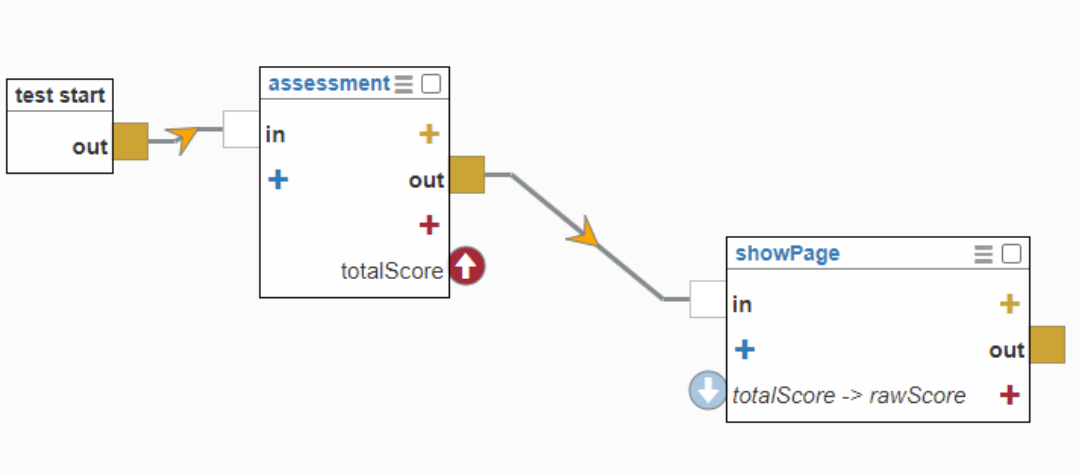
- Another way to accomplish this same thing would have been to set the output port of the assessment node and the input port of the showPage node to be ‘flow variable pointers’ and

give them the same name. That way you don't need to drag one port to the other to make the connection, but the input port will be set to the value of the output port that has the same name.





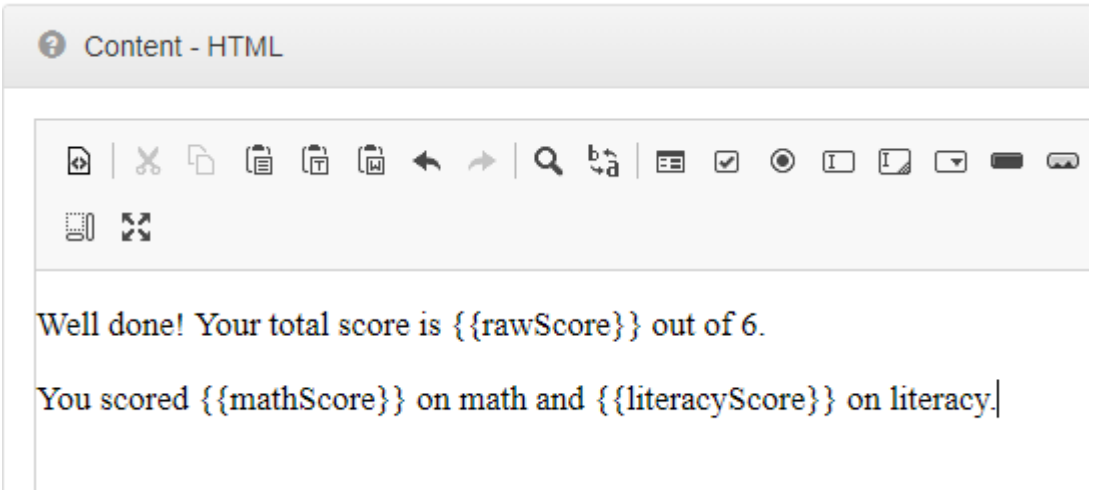
- Your flow would look like this if you did it that way. Using flow variable pointers can help to keep your test logic tidy in complex tests where you have many nodes and you don't want to be dragging connections all over the place. However it's good practice to keep naming consistent to avoid confusion. So in this example I would probably change my demoFeedback template to refer to {{totalScore}} rather than {{rawScore}}, or I could change the pointed variable name of totalScore on the assessment node to rawScore.



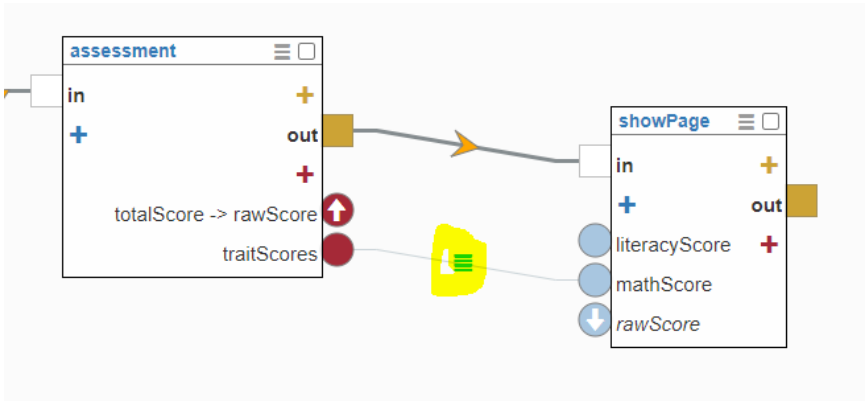
- You can use any native return ports of the assessment nodes in this way, passing them through to the showPage node and visualising them.

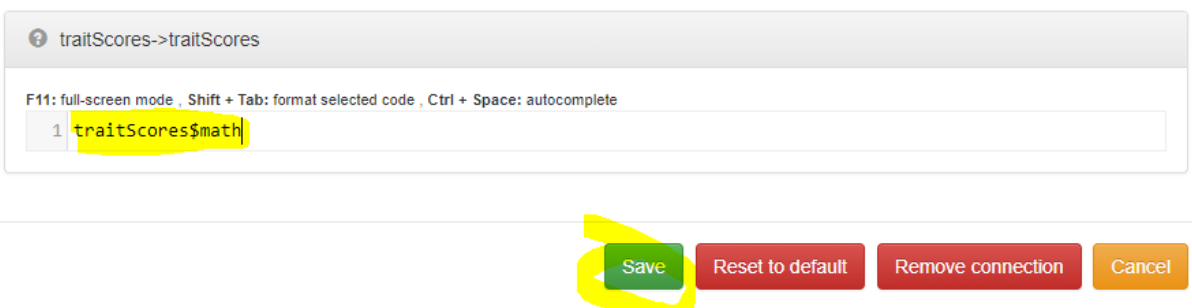
Multiple trait scores / filtering list objects

- Another native return of the assessment node is traitScores. This outputs an R list object containing key-value pairs. In my item bank I have the trait of each item assigned (math or literacy) so the assessment node will automatically calculate separate trait scores
- In this example I want to add trait scores to my feedback page, so I'll edit the content of my demoFeedback template to the following:

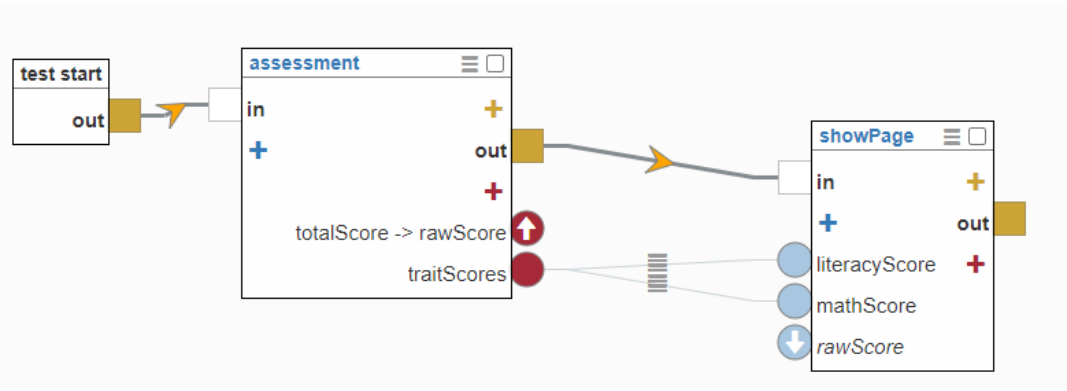


- Add two dynamic input ports to the showPage node and call them mathScore and literacyScore. To set the value of these ports, we need to take the traitScores list object and filter it so that we get only the trait score we want for each input. So I'll drag the traitScores output to the mathScore output, then click on the hamburger icon that appears and change its value to traitScores$math :





- Now do the same for the literacyScore, but this time set the value in the hamburger icon to traitScores$literacy. This tells the connection to only pass the value where trait equals math or literacy respectively. Your flow should look like this when done:



Now when you run the test you should get a feedback page that's something like this:

Well done! Your total score is 3 out of 6.

You scored 1 on math and 2 on literacy.

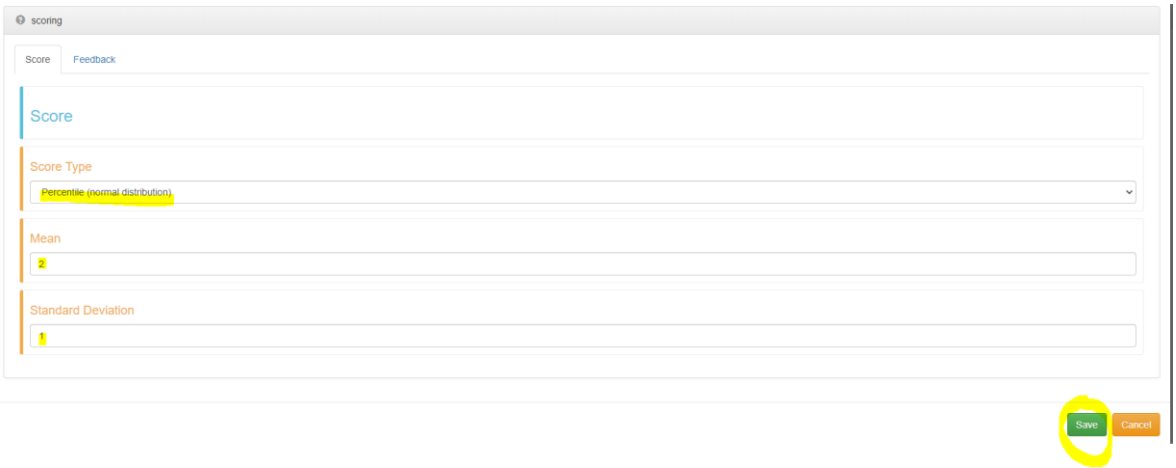
- Imagine you have a test that measures 10 traits. You can add a dynamic input to your showPage node for each trait score, and use these simple filters to obtain the value you need from the traitScores return port of the assessment node.

Scoring node: transforming scores and adding text interpretation

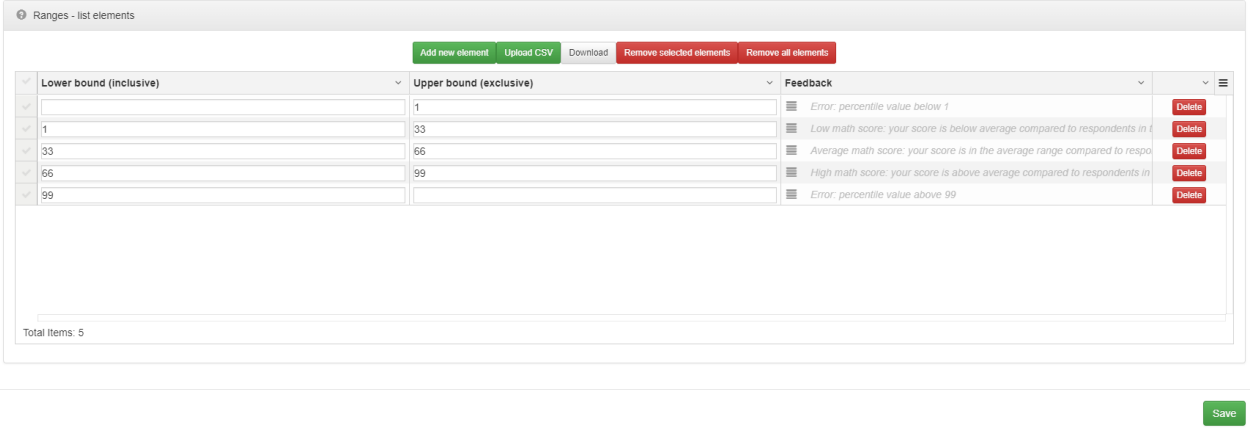
The scoring node has several popular score transformation algorithms built in and can be used to either transform outputs of the assessment node to new values, or to add interpretation based on score ranges, or both. You can see exactly how each of the transformations work by going to Tests -> starter content -> _scoring test -> Edit and looking at the R code comprising this node.

Imagine I know the norm (mean and standard deviation of each of the two scales in my test, and I want to add text interpretation based on percentile or stanine ranges. Each scoring node can only work with one input score at a time, so if your test has two traits you will need to use two separate scoring nodes. Follow this process:

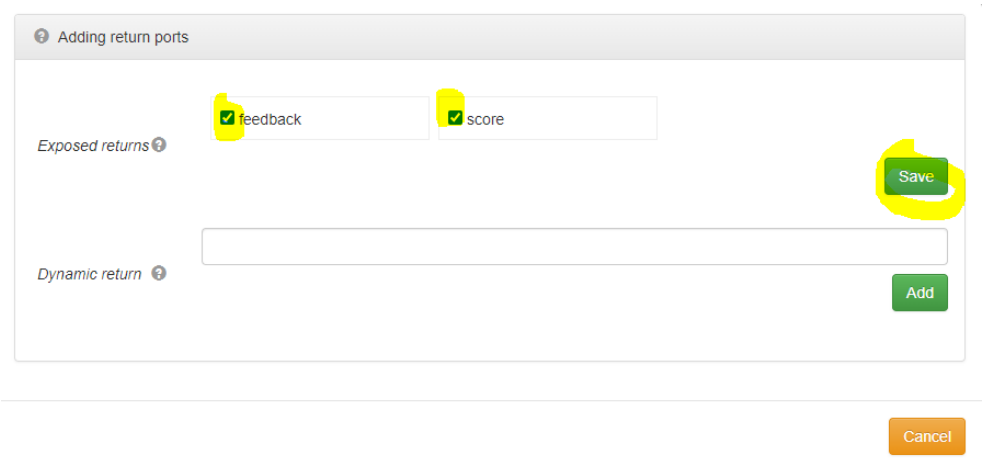
- Add a scoring node and expose the 'rawScore' input port. Drag the traitScores output from assessment node to this input port and filter it to just pass the math score, as we did above.
- In the scoring node wizard, select 'Percentile (normal distribution' as the Score type and enter the mean and SD (in this case 2 and 1 respectively).



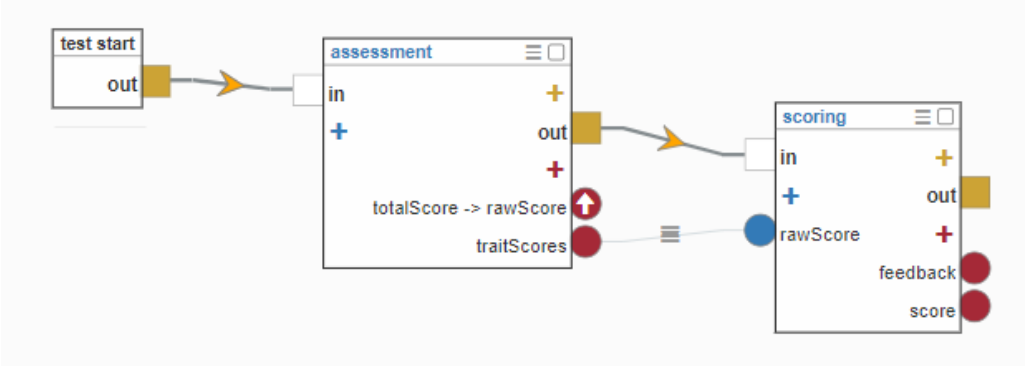
- Click the Feedback tab in the scoring node and select ranges. Click the edit hamburger icon and you will see a wizard to create a table containing conditional feedback that will be shown based on an evaluation of where the percentile score sits between lower and upper bounds. Click 'Add new element' fives times to add five new rows. It is a good idea to handle edge cases (i.e. what happens when something goes wrong with the scoring?). If you the lower bound or upper bound columns blank for a given row, then it will treat it as minus or plus infinity respectively. In this example I want to show feedback based on three ranges of percentiles, so my table will look like this. You can make your feedback as detailed as you like.:



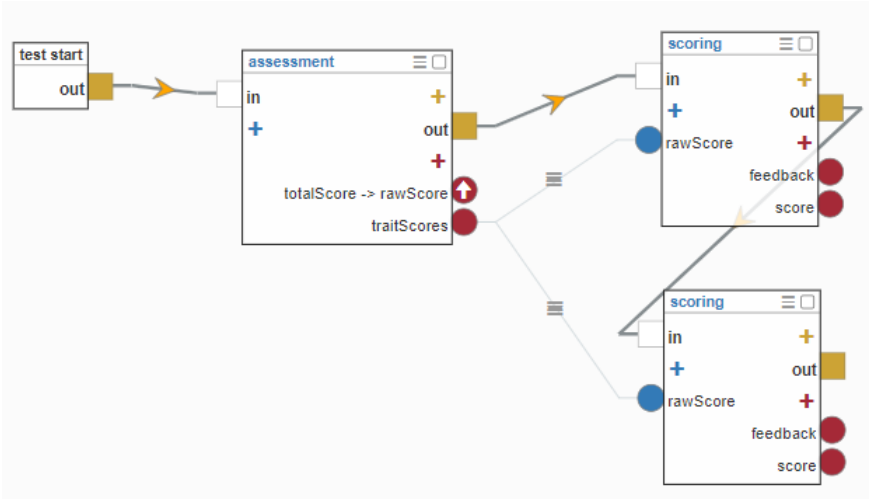
- Expose the score and feedback return ports of the scoring node, so we can use them elsewhere in the flow



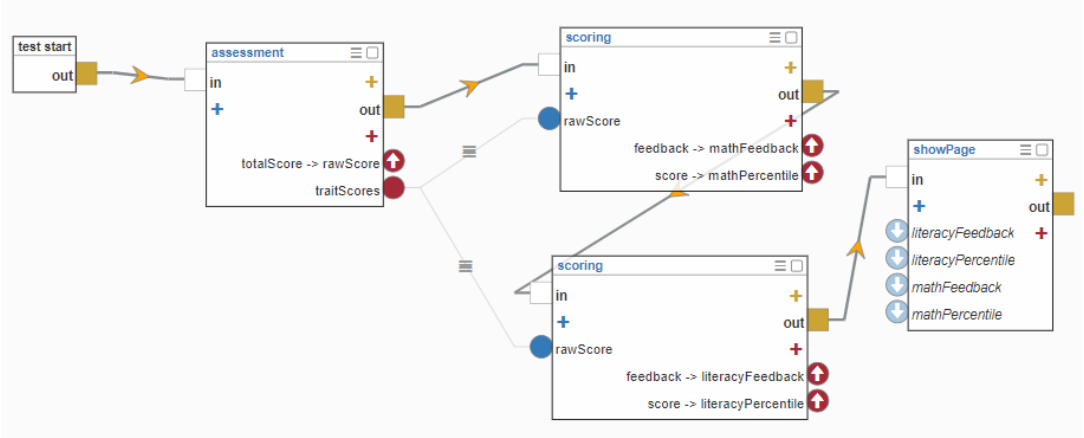
- Your flow should now look like this:



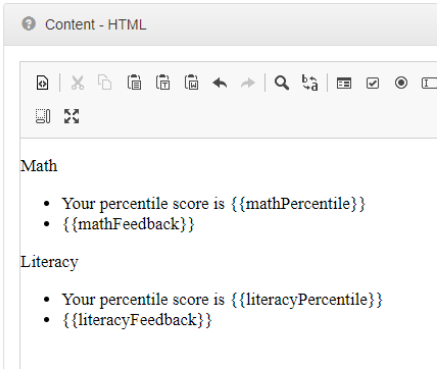
- Repeat this process to add a second scoring node that will take traitScores$literacy as an input and show feedback based on ranges of literacy scores. Connect the execution flow and it should all look like this when done:



- Now I need to pass all the variables I am going to need to refer to from my feedback page to the showPage node. I will do this using dynamic ports and flow variable pointers but you could also do it by dragging ports to create the relevant connections. This has been covered above. Your flow should look something like this when you're done:



- I'll now set the content of my demoFeedback template to the following, so it pulls through these input variables and shows them in the relevant places on the page:



- Now when you run the test and answer some questions, you will see the calculated percentile score and the corresponding feedback text shown on the feedback page:

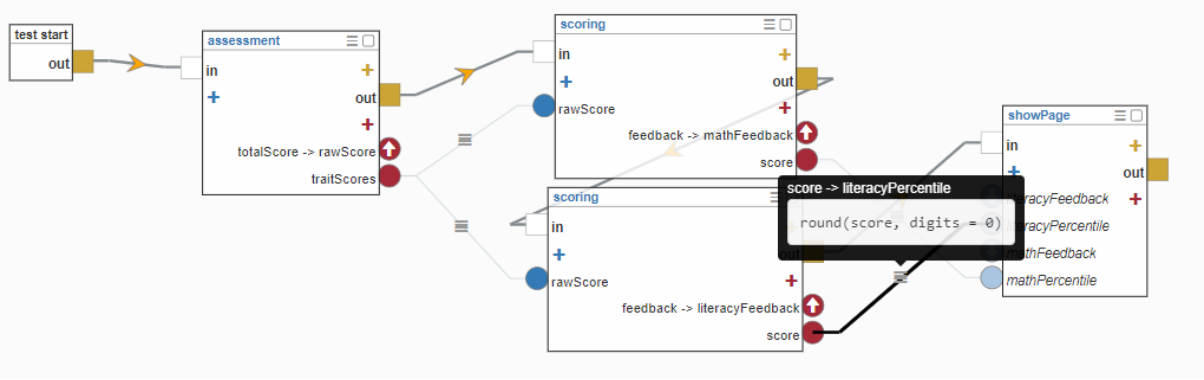
Math

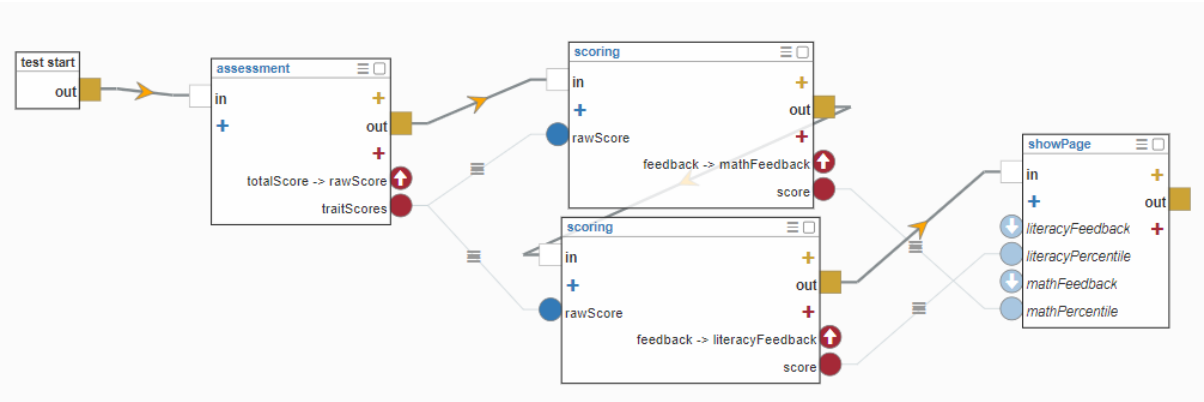
- Your percentile score is 50
- Average math score: your score is in the average range compared to respondents in the norm sample

Literacy

- Your percentile score is 15.8655253931457
- Low literacy score: your score is below average compared to respondents in the norm sample

- If you want to perform further functions on values before showing them on the page, you could use manual port connection (drag and drop) rather than flow variable pointers, and edit the value in the hamburger icon. Here is an example, where I round the percentile values to an integer before passing them to the showPage input port.





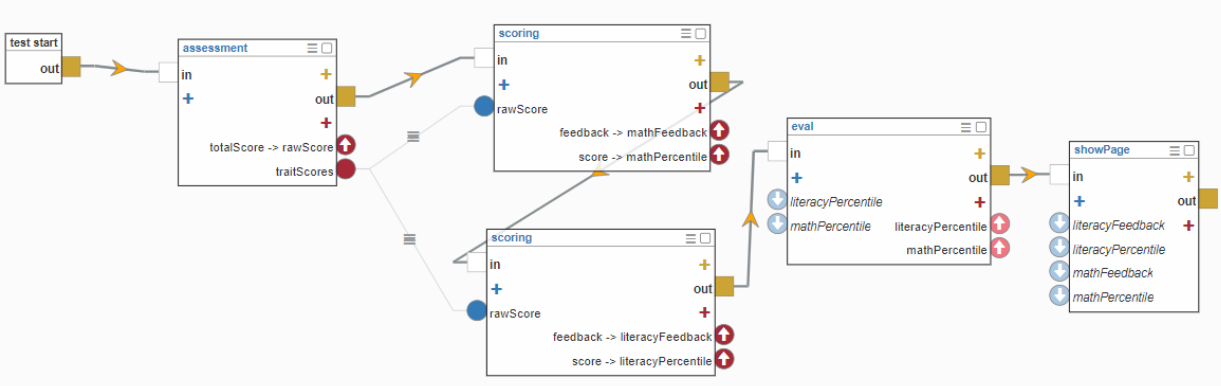
- Alternatively, you could add an eval node in between the scoring nodes and the showPage node, which will run this same R code to round the percentile values. In that case, my eval node would contain the following code:

Code - R code

F11: full-screen mode , **Shift + Tab:** format selected code , **Ctrl + Space:** autocomplete

```
1 mathPercentile <- round(mathPercentile, digits = 0)
2 literacyPercentile <- round(literacyPercentile, digits = 0)
```

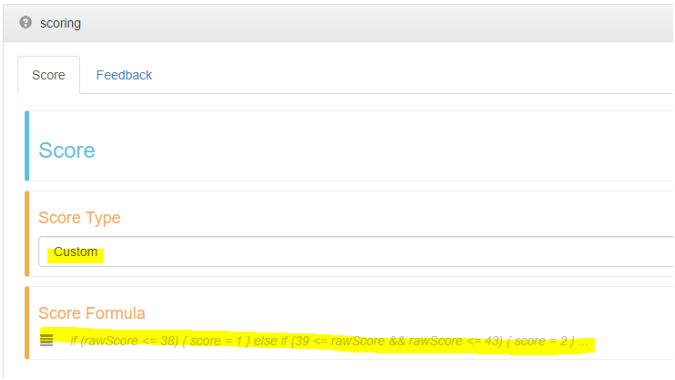
- And my flow (with the second approach) would look like this:



Custom scoring

You don't need to use the in-built score transformation algorithms in the scoring node when calculating feedback. You could write your score interpretation as R code and then pass the output variables to a showPage node to show custom feedback. This code could be in an eval node, or in the scoring node if you'd like to show text feedback based on the transformed score in a more convenient way (i.e. your text feedback can be kept in a table rather than in your code itself.

- Imagine that the math trait in my test has 100 questions rather than 3, and, rather than calculating percentiles, I want to convert the raw score on this trait to a stanine using a custom, cumulative frequency formula. I then want to show text feedback based on this stanine. To implement this, I would change the score type in my scoring node for the math trait to 'Custom' and then enter the R code in the 'Score Formula' field.



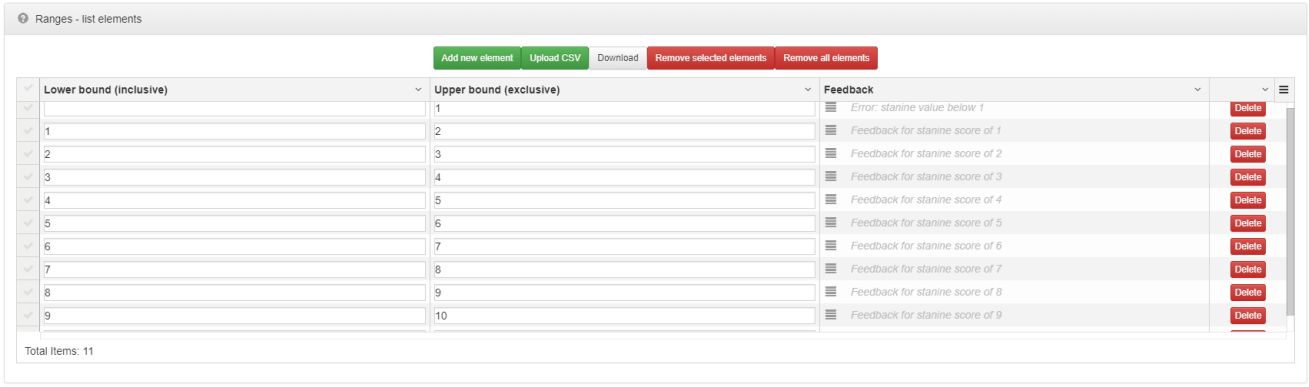
- My custom code looks like this. It takes rawScore as an input and applies some simple if else logic based on ranges of this rawScore to output a variable called 'score' which is a stanine from 1 to 9:

Score Formula - R code

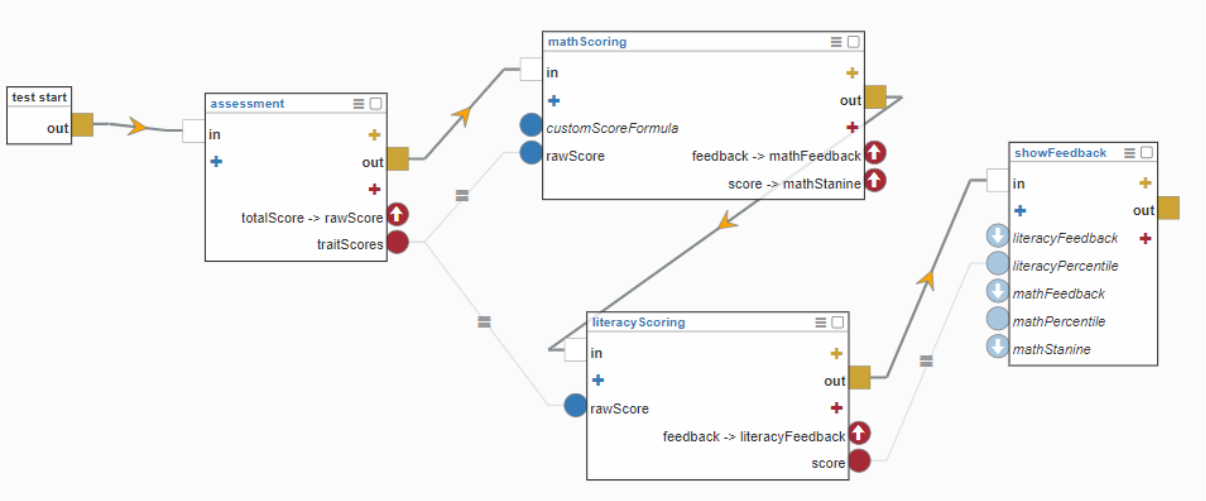
F11: full-screen mode , **Shift + Tab:** format selected code , **Ctrl + Space:** autocomplete

```
1  if (rawScore <= 38) {
2     score = 1
3  } else if (39 <= rawScore && rawScore <= 43) {
4     score = 2
5  } else if (44 <= rawScore && rawScore <= 48) {
6     score = 3
7  } else if (49 <= rawScore && rawScore <= 53) {
8     score = 4
9  } else if (54 <= rawScore && rawScore <= 58) {
10    score = 5
11 } else if (59 <= rawScore && rawScore <= 62) {
12    score = 6
13 } else if (63 <= rawScore && rawScore <= 66) {
14    score = 7
15 } else if (67 <= rawScore && rawScore <= 70) {
16    score = 8
17 } else if (71 <= rawScore) {
18    score = 9
19 }
20 print(score)
```

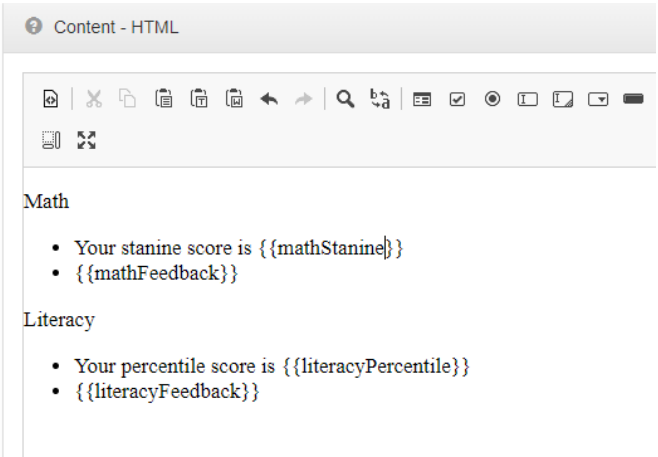
- I'll now change my feedback table to output text interpretation based on stanine score, and expose the customScoreFormula input port on the scoring node for convenience.



- Now I will add another dynamic input to my showPage node called mathStanine and set its value from my custom scoring node using flow variable pointers. My flow now looks like this. Note that I have renamed my nodes for clarity and the input port ‘mathPercentile’ is no longer being used:



- Lastly I’ll change the content of my demoFeedback template to the following:



Now when I run the test I will see something like this:

Math

- Your stanine score is 1
- Feedback for stanine score of 1

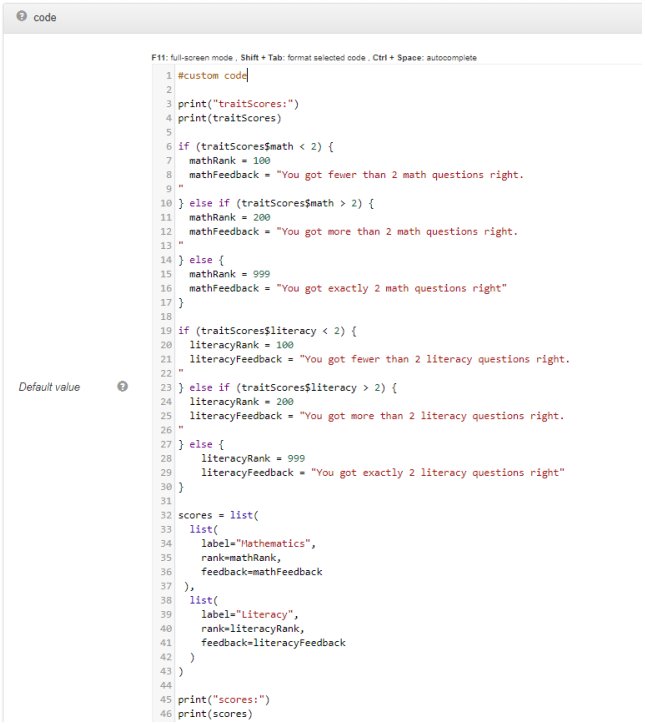
Literacy

- Your percentile score is 16
- Low literacy score: your score is below average compared to respondents in the norm sample

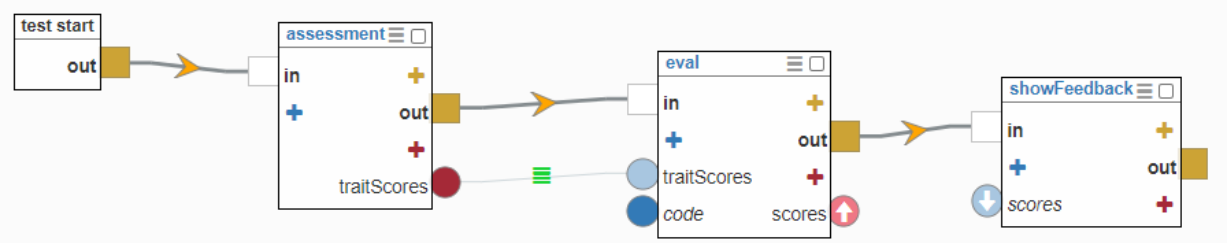
Using javascript and iterating feedback

Sometimes it's useful to access test variables from javascript rather than from R. You can make use of the Concerto testrunner to do this, by adding scope to the javascript section of the template that will need to use the variables. This can also be a useful approach when you need to repeat the same type of element many times on the page but you want to avoid duplication of code.

Imagine I have implemented some custom scoring to my test, which assigns so-called 'rank' values based on the number of questions answered correctly, and also shows some feedback. For the sake of this example I have put this custom scoring into an eval node, which takes an input called 'traitScores' and then produces an output called scores. This output is a list object containing label, rank and feedback for each of my two traits. The code in my eval node looks like this:



- I have set the scores output as a flow variable pointer and will pull it into my showPage node in the same way. My flow looks like this:
-



- Imagine that my test had 20 traits, each with a rank, label and feedback property. It would be rather time consuming to create dynamic inputs for every variable we want to be accessible from the showPage node, and then dragging and filtering 60 different connections from my previous nodes. The scores list object helps us access these properties more cleanly
- All I need to do is go into my demoFeedback template which this showPage node is using and add $scope.sores = testRunner.R.scores to the javascript section, as shown below.

javascript

F11: full-screen mode , **Shift + Tab**: format selected code

```
1 testRunner.controllerProvider.register("page", function($scope) {
2   $scope.buttonLabel = testRunner.R.buttonLabel;
3   $scope.scores = testRunner.R.scores;
4 });
5
```

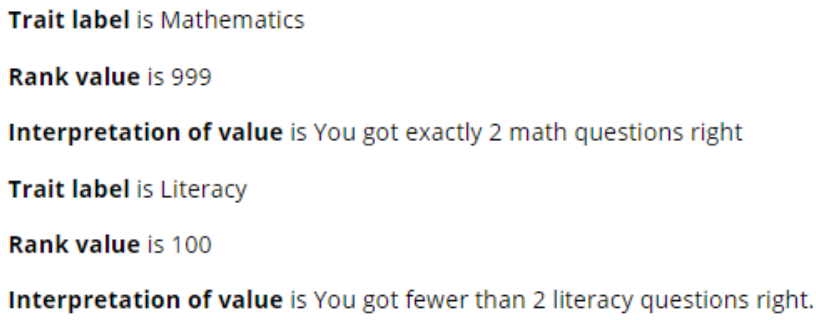
- Now the object 'scores' is accessible client-side when my demoFeedback template loads. I can refer it to variables in it in a similar way to how I do with variables in R, however this time we use [[square brackets]]. For example I could print the value of the mathRank variable by typing [[scores.mathRank]] in the content of my template. This tells the template to look in the javascript object called 'scores' for the property 'mathRank'.
- Another thing I can do is use the javascript function ng-repeat to have my template iterate through the scores object, finding the relevant properties by label, and then showing them one after the other. To do this I would replace the {{content}} placeholder with the following in the HTML of my demoFeedback template. I've added some inline styling to make some parts bold but this is just decorative:

```
<div class="questions">
    <div class="container">
        <div ng-repeat="scores in scores">
            <p>
                <strong>
                    Trait label
                </strong>
                is [[scores.label]]
            </p>

            <p>
                <strong>
                    Rank value
                </strong>
                is [[scores.rank]]
            </p>

            <p>
                <strong>
                    Interpretation of value
                </strong>
                is [[scores.feedback]]
            </p>
        </div>
    </div>
```

- Now when you run the test, the template will automatically iterate to print all the values in the scores object, and you should see something like this:



- By adding <div ng-repeat=”scores in scores”> I am telling the template that within that div element, I can refer to javascript properties that are contained in the ‘scores’ object. Without this function, if I were passing variables one by one from R for example, then I would need to type something like the following in order to achieve the same end result in terms of visualisation:

Trait label is Mathematics

Rank value is {{mathRank}}

Interpretation of value is {{mathFeedback}}

Trait label is Literacy

Rank value is {{literacyRank}}

Interpretation of value is {{literacyFeedback}}

- You can imagine you would have a lot of code duplication for tests with many traits where your feedback layout is repetitive, so using the iteration can be more efficient.