# Looping Functions

Work through the following exercises to help develop your skills using looping functions. Read each exercise instruction carefully. For exercises that don’t require the creation of a new rule, simply create a new expression rule called **testRule** and use it to perform the tasks listed.

**Exercise 1: Create a list of Skills and their ratings**

Use the apply, merge, and reduce functions to create a formatted list of skills. If the list of skills contains: Leader, Hard-Working, and Organized and the corresponding ratings are 3;5;2, then the list should be displayed as follows:

Leader: 3

Hard-Working: 5

Organized: 2

Step-by-Step

1. Create an expression rule called **D202\_Ex4\_1**
2. The rule should have two inputs: “categories” (text array) and “ratings” (integer array)
3. Use the apply, concat, and merge functions to join the two lists together.
   1. Between each skill, you should also concatenate “:” which can be achieved by using the repeat function.
   2. After each rating, you should also concatenate a carriage return using the the repeat function. *Hint: A carriage return is represented by the expression char(10).*
   3. In order to concatenate the objects in the correct order, use fn!concat as the array function and merge to combine the lists together.
4. After joining the lists of skills and ratings, combine all of the arrays into a single function using the reduce function.
   1. The reduce function should also use fn!concat to combine all lists together.

*Review the Skills practice solutions sheet for the correct result.*

**Exercise 2: Using nested apply() statements**

In this exercise, create a form that allows users to input their skills. The skills are as follows: Punctual, Organized, Trustworthy, and Creative. Each of the skills should allow an input for an integer value from 1-5, and they should include radio buttons to allow users to select their proficiency in each skill.

Once all of the skills are selected, icons should be shown corresponding to each skill, with the color of the icon designating whether the skill is high, medium, or low. If the value is greater than 3, the skill image should be green; if it is less than 3, the skill image should be red; if it equals 3, the skill image should be grey. Here are the skills and their corresponding images from the a!iconNewsEvent function:

* Punctual “CLOCK”
* Organized “FILE\_CABINET”
* Trustworthy “HANDSHAKE”
* Creative “MUSIC\_NOTE”

Step-by-step

1. Create a constant called **D202\_SKILL\_IMAGES** to keep all of the icon text values.
   1. Make sure the constant is an array and add the following values: “CLOCK”;”FILE\_CABINET”;””HANDSHAKE”;”MUSIC\_NOTE” *Note: The case of the values must exactly match the values above.*
2. Create a supporting rule to test whether each value is less than 3, equal to 3, or greater than 3.
   1. Create a new expression rule called **D202\_ColorRating** and add a rule input for “value”
   2. Use nested if() statements to check the value. Less than 3 should return “RED”; equal to 3 should return “GREY”; greater than 3 should return “GREEN”.
3. Create a new interface called **D202\_Ex4\_2** and add inputs for “punctual”, “organized”, “trustworthy”, and “creative” (all should be integers).
   1. Create a form layout within the interface and define radio button fields for each of the skills. Configure for each radio button field:
      1. Set the label to the name of the skill
      2. Save the value into the corresponding rule input.
      3. Define the choice labels and choice values to use values from 1-5. *Hint: You can either hard-code values for 1-5, use the enumerate function, or use a!radioButtonFieldByIndex().*
4. Add an image field to display the list of documents. This field will require nesting of several array functions. *Hint: It is easiest to build from the top level array first. Find which data type is expected within that array, and keep going until you get down to the final components. Also, if you are having trouble getting the image field to work correctly, try in another interface first and use an array of ratings such as {1,5,2,3}.*
   1. Start by using apply() with the a!documentImage() function. This will allow you to display multiple document images.
   2. The documents used in the a!documentImage() function can be built using another apply function. This apply() should use the a!iconNewsEvent function.
      1. The a!iconNewsEvent function is expecting two inputs: the name of the icon and the color. To add two inputs to this apply function, you must use a merge() to merge the two lists together.
      2. The first input of the merge will come from your constant D202\_SKILL\_IMAGES.
      3. The second input will be determined using your rule for D202\_ColorRating. Since this is a rule applied multiple times, the list of colors must be defined with one more apply() function. Use the rule shown as the first parameter of the array and add all of the rule inputs into an array for the second parameter using brackets {}.
5. Once you create the images, add a validation for the form to make sure a value is provided for each skill. Instead of adding a validation for each skill individually, add a validation for the form to make sure all fields have a value.
   1. Use an if() statement to define the validation. The error message should display, “Please make sure all skill ratings have been entered”.
   2. Use any() to check if any of the fields have a null value.