DYNAMIC FILTERING

If you let users filter the contents of a page, you can build all of the HTML content, and then show and hide the relevant parts as the user interacts with the filters.

Imagine that you were going to provide the user with a slider so that they could update the price that they were prepared to pay per hour. That slider would automatically update the contents of the table based upon the price range the user had specified.

If you built a new table every time the user interacts with the slider (like the previous two examples that showed filtering), it would involve creating and deleting a lot of elements. Too much of this type of DOM manipulation can slow down your scripts.

A far more efficient solution would be to:

1. Create a table row for *every* person.
2. Show the rows for the people that are within the specified range, and hide the rows that are outside the specified bounds.

Below, the range slider used is a jQuery plugin called noUiSlider (written by Léon Gerson).
`http://refreshless.com/nouislider/`

Before you see the code for this example, take a moment to think about how to approach this script... Here are the tasks that the script needs to perform:

i) It needs to go through each object in the array and create a row for that person.

ii) Once the rows have been created, they need to be added to the table.

iii) Each row needs to be shown / hidden depending on whether that person is within the price range shown on the slider. (This task happens each time the slider is updated.)

In order to decide which rows to show / hide, the code needs to cross-reference between:

- The `person` object in the `people` array (to check how much that person charges)
- The row that corresponds to that person in the table (which needs to be made visible or hidden)

To build this cross-reference we can create a new array called `rows`. It will hold a series of objects with two properties:

- `person`: a reference to the object for this person in the `people` array
- `$element`: a jQuery collection containing the corresponding row in the table

In the code, we create a function to represent each of the tasks identified on the left. The new cross-reference array will be created in the first function:

`makeRows()` will create a row in the table for each person *and* add the new object into the `rows` array

`appendRows()` loops through the `rows` array and adds each of the rows to the table

`update()` will determine which rows are shown or hidden based on data taken from the slider

In addition, we will add a fourth function: `init()` This function contains all of the information that needs to run when the page first loads (including creating the slider using the plugin).

`init` is short for **initialize**; you will often see programmers using this name for functions or scripts that run when the page first loads.

Before looking at the script in detail, the next two pages are going to explain a little more about the `rows` array and how it creates the cross-reference between the objects and the rows that represent each person.

STORING REFERENCES TO OBJECTS & DOM NODES

The **rows** array contains objects with two properties, which associate:
1: References to the objects that represent people in the **people** array
2: References to the row for those people in the table (jQuery collections)

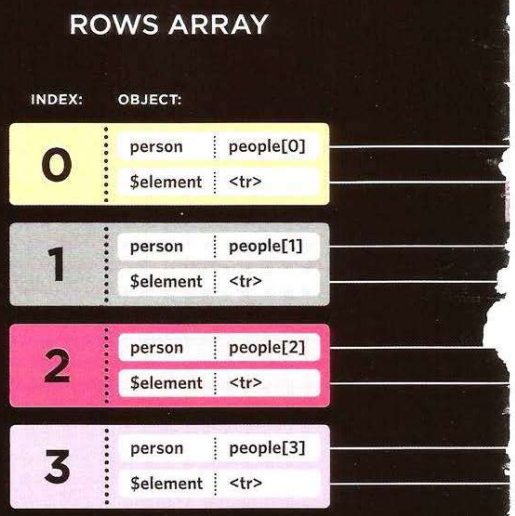
You have seen examples in this book where variables were used to store a reference to a DOM node or jQuery selection (rather than making the same selection twice). This is known as **caching**.

This example takes that idea further: as the code loops through each object in the **people** array creating a row in the table for that person, it also creates a new object for that person and adds it to an array called **rows**. Its purpose is to create an association between:

- The object for that person in the source data
- The row for that person in the table

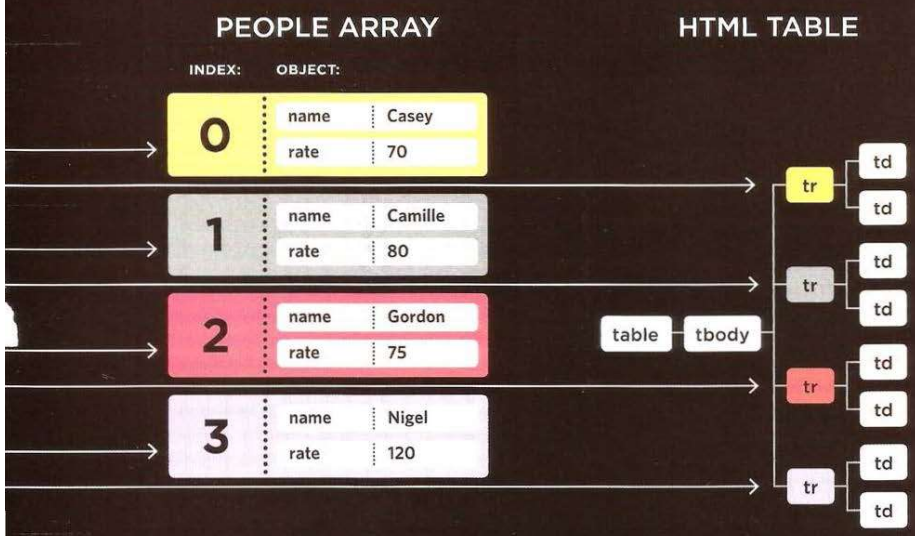
When deciding which rows to show, the code can then loop through this new array checking the person's rate. If they are affordable, it can show the row. If not, it can hide the row.

This takes less resources than recreating the contents of the table when the user changes the rate they are willing to pay.



On the right, you can see the **Array** object's **push()** method creates a new entry in the **rows** array. The entry is an object literal, and it stores the **person** object and the row being created for it in the table.

```
rows.push({
  person: this,    // person object
  $element: $row   // jQuery collection
});
```

The **people** array already holds information about each person and the rates that they charge, so the object in the **rows** array only needs to point to the original object for that person (it does not copy it).

A jQuery object was used to create each row of the table. The objects in the **rows** array store a reference to each individual row of the table. There is no need to select or create the row again.

DYNAMIC FILTERING

1. Place the script in an IIFE (not shown in flowchart). The IIFE starts with the `people` array.
2. Next, four global variables are created as they are used throughout the script:
`rows` holds the cross-referencing array .
`$min` holds the input to show the minimum rate.
`$max` holds the input to show the maximum rate.
`$table` holds the table for the results.

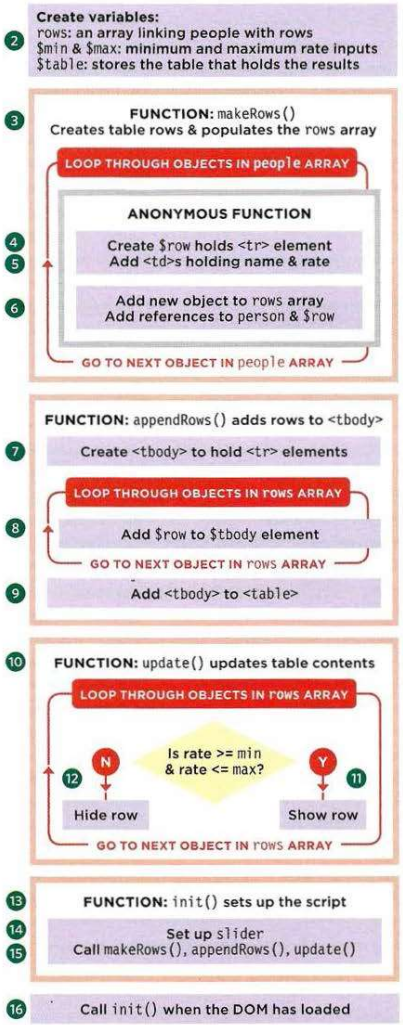
3. `makeRows()` loops through each person in the `people` array calling an anonymous function for each object in the array. Note how `person` is used as a parameter name. This means that within the function, `person` refers to the current object in the array.
4. For each person, a new jQuery object called `$row` is created containing a `<tr>` element.
5. The person's name and rate are added in `<td>`s.
6. A new object with two properties is added to the `rows` array: `person` stores a reference to their object, `$element` stores a reference to their `<tr>` element.

7. `appendRows()` creates a new jQuery object called `$tbody` containing a `<tbody>` element.
8. It then loops through all of the objects in the `rows` array and adds their `<tr>` element to `$tbody`.
9. The new `$tbody` selection is added to the `<table>`.

10. `update()` goes through each of the objects in the `rows` array and checks if the rate that the person charges is more than the minimum and less than the maximum rate shown on the slider.
11. If it is, jQuery's `show()` method shows the row.
12. If not, jQuery's `hide()` method hides the row.

13. `init()` starts by creating the slide control.
14. Every time the slider is changed, the `update()` function is called again.
15. Once the slider has been set up, the `makeRows()`, `appendRows()`, `update()` functions are called.

16. The `init()` function is called (which will in turn call the other code).



FILTERING AN ARRAY

JAVASCRIPT c12/js/dynamic-filter.js

```
(function(){                              // PEOPLE ARRAY GOES HERE
  var rows = [],                          // rows array
      $min = $('#value-min'),             // Minimum text input
      $max = $('#value-max'),             // Maximum text input
      $table = $('#rates');               // The table that shows results
  function makeRows() {                   // Create table rows and the array
    people.forEach(function(person) {     // For each person object in people
      var $row = $('<tr></tr>');          // Create a row for them
      $row.append( $('<td></td>').text(person.name) ); // Add their name
      $row.append( $('<td></td>').text(person.rate) ); // Add their rate
      rows.push({ // Add object to cross-references between people and rows
        person: person,                   // Reference to the person object
        $element: $row                    // Reference to row as jQuery selection
      });
    });
  }
  function appendRows() {                 // Adds rows to the table
    var $tbody = $('<tbody></tbody>');    // Create <tbody> element
    rows.forEach(function(row) {          // For each object in the rows array
      $tbody.append(row.$element);        // Add the HTML for the row
    });
    $table.append($tbody);                // Add the rows to the table
  }
  function update(min, max) {             // Update the table content
    rows.forEach(function(row) {          // For each row in the rows array
      if (row.person.rate >= min && row.person.rate <= max) { // If in range
        row.$element.show();              // Show the row
      } else {                            // Otherwise
        row.$element.hide();              // Hide the row
      }
    });
  }
  function init() {                       // Tasks when script first runs
    $('#slider').noUiSlider({             // Set up the slide control
      range: [0, 150], start: [65, 90], handles: 2, margin: 20, connect: true,
      serialization: { to: [$min,$max], resolution: 1 }
    }).change(function() { update($min.val(), $max.val()); });
    makeRows();                           // Create table rows and rows array
    appendRows();                         // Add the rows to the table
    update($min.val(), $max.val());       // Update table to show matches
  }
  $(init);                                // Call init() when DOM is ready
}());
```