Riverside JS Workshop
JavaScript Skills Ladder, Lesson 2.8

Implementing Local Storage

Lesson Objectives

When you complete this lesson, you will be able to:

- *store objcts in local storage.*
- *save to-do items in local storage.*
- *add an ID to a to-do item.*
- *get to-do items from local storage.*
- *use the substring() method to take any String and create a new String from it.*

Storing Objects in Local Storage?

In the previous lesson, you learned the basics of storing data in the browser, using Local Storage. In this lesson, we're going update our To-Do List application to store to-do items in Local Storage, rather than on the server.

You know how to store simple strings in Local Storage, and you know that you can store only strings in Local Storage (meaning you can't store numbers, objects, or other types in Local Storage). A to-do item is an object. Here's an example of one:

OBSERVE:

```
{
  task: "get milk",
  who: "Scott",
  dueDate: "today",
  done: false
}
```

Knowing that you can only store strings in Local Storage, can you think of how we might store to-do items? And what should we use as the keys for the to-do items?

We can use JSON to store objects such as our to-do items in Local Storage. Recall that JSON allows us to represent JavaScript objects as strings. Once we have a string representation of an object, we can store it in Local Storage just like any other string. Similarly, we can use JSON to retrieve the string from Local Storage and turn it back into an object.

Before we dive back into the To-Do List application, let's create a small example to store and retrieve JSON objects from Local Storage. Create a new HTML file as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
    <title>Store an Object in Local Storage</title>
    <meta charset="utf-8">
    <script src="storeObj.js"></script>
</head>
<body>
<h1>Store an object in Local Storage</h1>
</body>
</html>
```

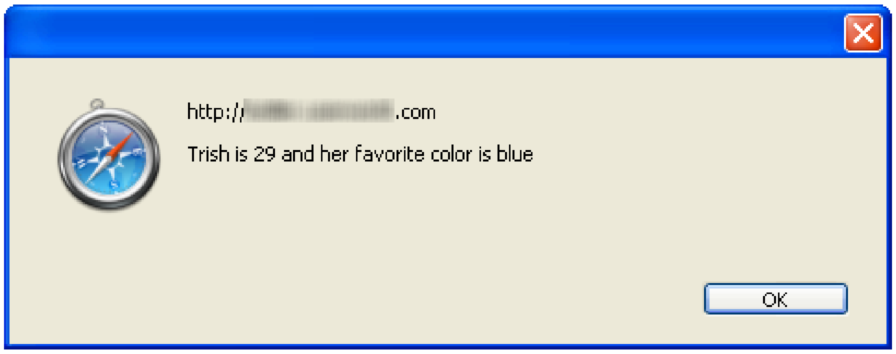
Save this file in a work folder as *storeObj.html*. We link to the file *storeObj.js* in the header—that's where all the real action is! Now we need some HTML so we can preview and look Local Storage using the browser developer console.

Create a new file and add the JavaScript as shown:

CODE TO TYPE:

```
window.onload = init;

function init() {
    var myObject = {
        name: "Trish",
        age: 29,
        favColor: "blue"
    };
    var myObjectJson = JSON.stringify(myObject);
    localStorage.setItem("trish", myObjectJson);
    var newMyObjectJSON = localStorage.getItem("trish");
    var newMyObject = JSON.parse(newMyObjectJSON);
    alert(newMyObject.name + " is " + newMyObject.age +
        " and her favorite color is " + newMyObject.favColor);
}
```

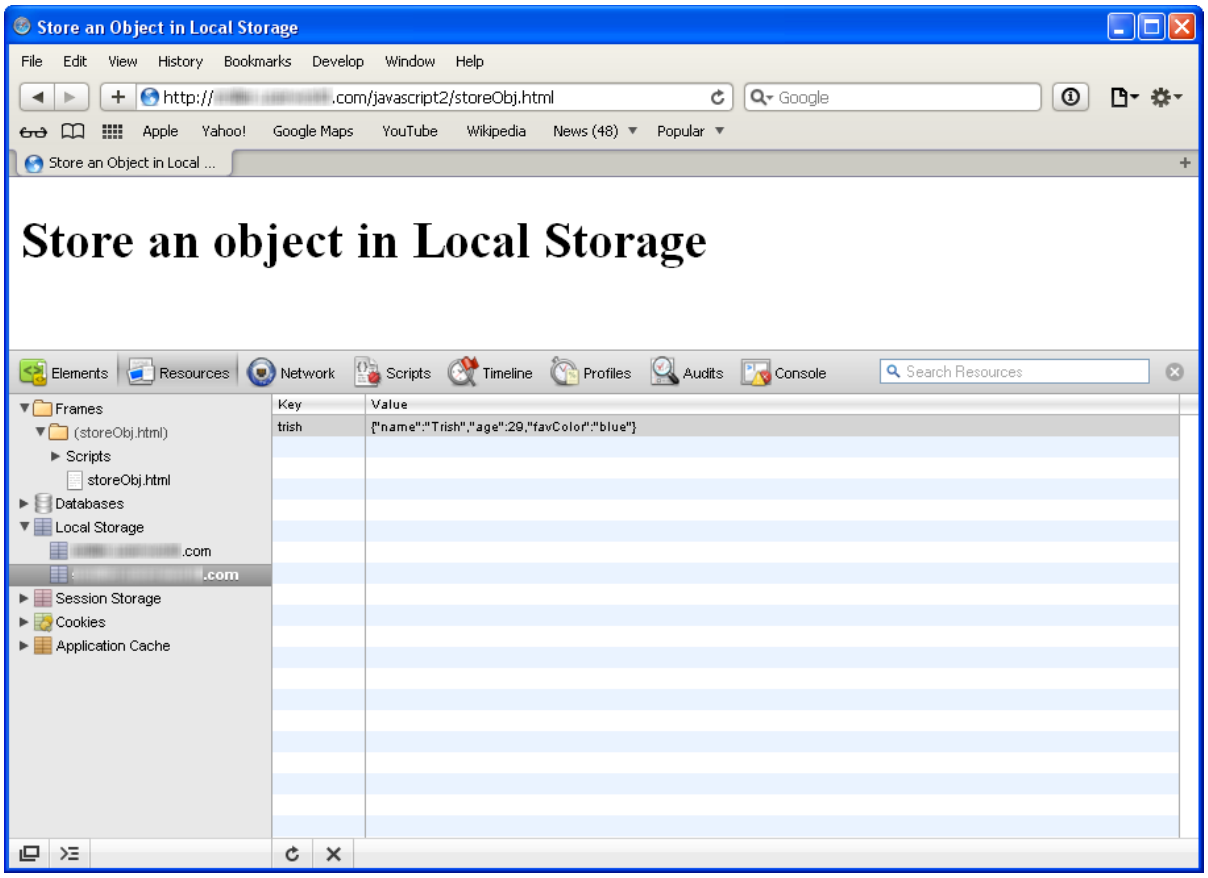
Save this file in the same work folder as *storeObj.js*. Now open your *storeObj.html* file in a browser. You see an alert like this:



http://[redacted].com

Trish is 29 and her favorite color is blue

Use your browser's developer console to inspect Local Storage. The Local Storage includes the item you just stored using this program:



http://[redacted].com/javascript2/storeObj.html

Store an object in Local Storage

Local Storage: [redacted].com, [redacted].com

Key	Value
trish	{"name":"Trish","age":29,"favColor":"blue"}

Note You might see additional items in Local Storage that you've added while completing other lessons. That’s fine. Just make sure you see the "trish" object you just added!

Let's look at the JavaScript:

OBSERVE:

```
function init() {
    var myObject = {
        name: "Trish",
        age: 29,
        favColor: "blue"
    };
    var myObjectJson = JSON.stringify(myObject);
    localStorage.setItem("trish", myObjectJson);
    var newMyObjectJSON = localStorage.getItem("trish");
    var newMyObject = JSON.parse(newMyObjectJSON);
    alert(newMyObject.name + " is " + newMyObject.age +
        " and her favorite color is " + newMyObject.favColor);
}
```

This `init()` function:

- creates an object named myObject.
- converts that object to JSON (a string) using the JSON object's `stringify()` method.
- stores the JSON version of the object in Local Storage, with the key "trish", using the `localStorage` object's `setItem()` method.
- retrieves a JSON string from Local Storage using the same key, "trish," using the `localStorage` object's `getItem()` method, and saving it in a new variable, `newMyObjectJSON`.
- creates `newMyObject` from that JSON string using the JSON object's `parse()` method.
- displays the values in `newMyObject` using dot notation, and alerts the values so you can see them (`newMyObject` has exactly the same structure as our original `myObject`, so we can do this).

So now that you know how to store and retrieve an object from Local Storage, let's go back to the To-Do List application and modify it to use Local Storage instead of Ajax to store to-do items.

Saving To-Do Items in Local Storage

Even though we use keys that relate to the content of the items we're storing (for example, "favGenre" and "browserWidth"), it's still difficult to figure out which items in Local Storage belong to which application. So, unless you use a string in your key that indicates that a particular item belongs to a specific application, how will your application know which items it can use? Remember that more than one application can use Local Storage at any given domain, and all these applications will be mixed together in Local Storage; we need a way to identify the items that belong to the To-Do List application.

For now, let's create a key for each to-do item from a unique id and a string that represents the application. We'll create the unique id for each to-do item as that item is created (that is, when you submit a new to-do item using the To-Do List form), store that id (along with the other to-do list information) in the Todo object, and then use that id when we create the key to store the item in Local Storage. Let's update the code and then we'll go over each change.

Note We'll continue to update the *todo.html* and *todo.js* files. You might want to make copies of the files to save your previous work.

Modify *todo.js* as shown:

CODE TO TYPE:

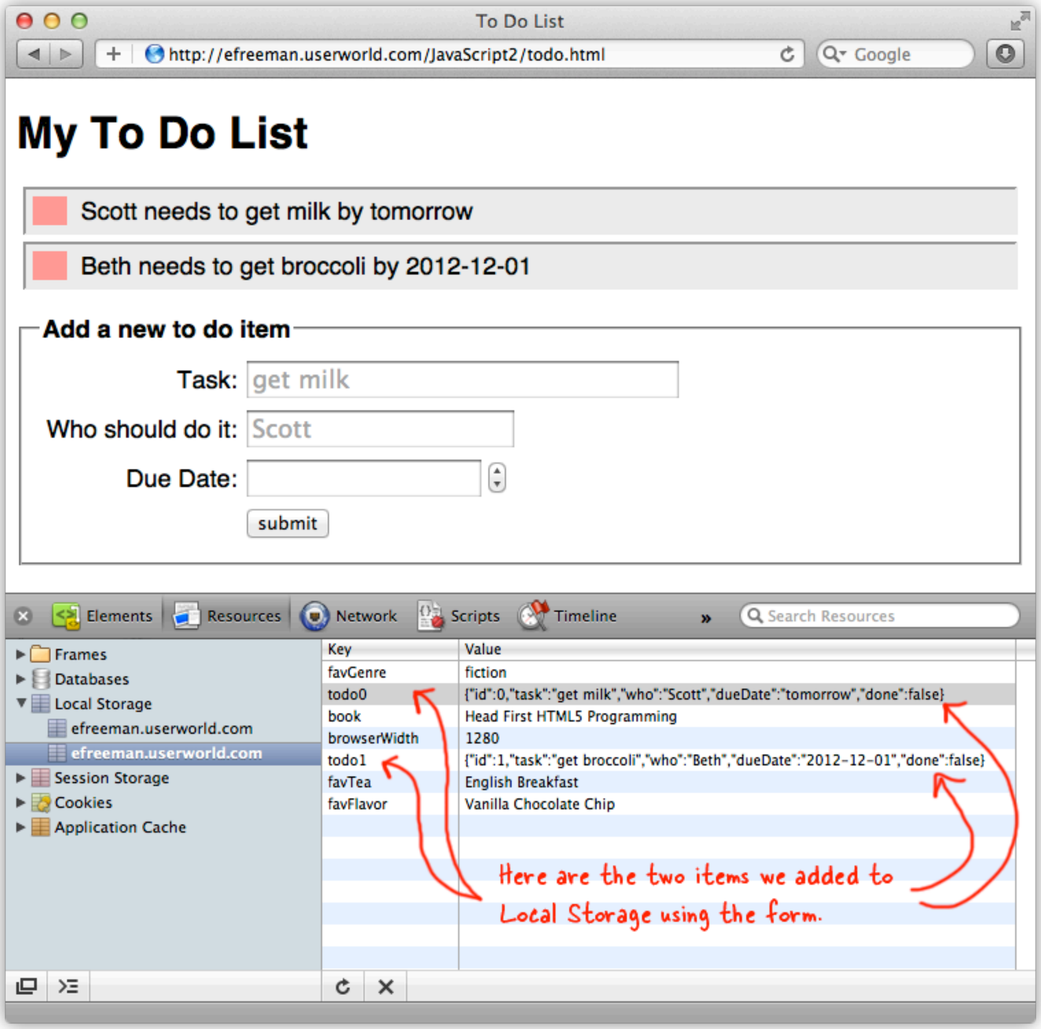
```
function Todo(id, task, who, dueDate) {
    this.id = id;
    this.task = task;
    this.who = who;
    this.dueDate = dueDate;
    this.done = false;
}

var todos = new Array();

window.onload = init;

function init() {
    var submitButton = document.getElementById("submit");
    submitButton.onclick = getFormData;

    ~~getTodoData();~~
```

```
}

~~function getTodoData() {~~
    ~~var request = new XMLHttpRequest();~~
    ~~request.open("GET", "todo.json");~~
    ~~request.onreadystatechange = function() {~~
        ~~if (this.readyState == this.DONE && this.status == 200) {~~
            ~~if (this.responseText) {~~
                ~~parseTodoItems(this.responseText);~~
                ~~addTodosToPage();~~
            ~~}~~
            ~~else {~~
                ~~console.log("Error: Data is empty");~~
            ~~}~~
        ~~}~~
    ~~};~~
    ~~request.send();~~
~~}~~

function parseTodoItems(todoJSON) {
    if (todoJSON == null || todoJSON.trim() == "") {
        return;
    }
    var todoArray = JSON.parse(todoJSON);
    if (todoArray.length == 0) {
        console.log("Error: the to-do list array is empty!");
        return;
    }
    for (var i = 0; i < todoArray.length; i++) {
        var todoItem = todoArray[i];
        todos.push(todoItem);
    }
}
function addTodosToPage() {
    var ul = document.getElementById("todoList");
    var listFragment = document.createDocumentFragment();
    for (var i = 0; i < todos.length; i++) {
        var todoItem = todos[i];
        var li = createNewTodo(todoItem);
        listFragment.appendChild(li);
    }
    ul.appendChild(listFragment);
}
function addTodoToPage(todoItem) {
    var ul = document.getElementById("todoList");
    var li = createNewTodo(todoItem);
    ul.appendChild(li);
    document.forms[0].reset();
}

function createNewTodo(todoItem) {
    var li = document.createElement("li");
    var spanTodo = document.createElement("span");
    spanTodo.innerHTML =
        todoItem.who + " needs to " + todoItem.task + " by " +
todoItem.dueDate;

    var spanDone = document.createElement("span");
    if (!todoItem.done) {
        spanDone.setAttribute("class", "notDone");
        spanDone.innerHTML = "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
    }
    else {
        spanDone.setAttribute("class", "done");
        spanDone.innerHTML = "&nbsp;&#10004;&nbsp;";
    }

    li.appendChild(spanDone);
    li.appendChild(spanTodo);
    return li;
}

function getFormData() {
    var task = document.getElementById("task").value;
    if (checkInputText(task, "Please enter a task")) return;

    var who = document.getElementById("who").value;
    if (checkInputText(who, "Please enter a person to do the task"))
return;

    var date = document.getElementById("dueDate").value;
    if (checkInputText(date, "Please enter a due date")) return;

    var id = todos.length;
    var todoItem = new Todo(id, task, who, date);
    todos.push(todoItem);
    addTodoToPage(todoItem);
    ~~saveTodoData();~~
    saveTodoItem(todoItem);
}

function checkInputText(value, msg) {
    if (value == null || value == "") {
        alert(msg);
        return true;
    }
    return false;
}

function saveTodoItem(todoItem) {
    if (localStorage) {
        var key = "todo" + todoItem.id;
        var item = JSON.stringify(todoItem);
        localStorage.setItem(key, item);
    }
    else {
        console.log("Error: you don't have localStorage!");
    }
}
```

```
~~function saveTodoData() {~~
~~    var todoJSON = JSON.stringify(todos);~~
~~    var request = new XMLHttpRequest();~~
~~    var URL = "save.php?data=" + encodeURI(todoJSON);~~
~~    request.open("GET", URL);~~
~~    request.setRequestHeader("Content-Type",~~
~~                             "text/plain;charset=UTF-8");~~
~~    request.send();~~
~~}~~
```

Save these changes in *todo.js*. We removed the functions to save and get the to-do items (`saveTodoData()` and `getTodoData()`) from the server. We replaced the function to save the data with a new function, `saveTodoItem()`, which takes one to-do item and saves it to Local Storage.

Because we are still using the same JSON format we used before (in the Ajax version), we don't have to change much of the other code. That's convenient. Before we walk through the details, load the *todo.html* file in your browser. Try adding a few to-do items using the form:



My To Do List

🟥 Scott needs to get milk by tomorrow

🟥 Beth needs to get broccoli by 2012-12-01

Add a new to do item

Task:	get milk
Who should do it:	Scott
Due Date:	

submit

Key	Value
favGenre	fiction
todo0	{"id":0,"task":"get milk","who":"Scott","dueDate":"tomorrow","done":false}
book	Head First HTML5 Programming
browserWidth	1280
todo1	{"id":1,"task":"get broccoli","who":"Beth","dueDate":"2012-12-01","done":false}
favTea	English Breakfast
favFlavor	Vanilla Chocolate Chip

Here are the two items we added to Local Storage using the form.

When you look at Local Storage using your browser, you see the new items you added. If you had other items in Local Storage already, the new to-do items will be mixed in with those other items. You'll be able to tell which items belong to the To-Do List application because the keys all begin with "todo."

Right now we are *saving* the items we enter using the form in Local Storage (we're not retrieving those items when we first load the page), so you'll see new items being added to Local Storage and to the page as you add them using the form. If you reload the page, you won't see those items in your list yet. We'll get to that shortly.

Adding an ID to a Todo Item

We had to add an id property to the *Todo* object, so each to-do item could store a unique id:

OBSERVE:

```
function Todo(id, task, who, dueDate) {
    this.id = id;
    this.task = task;
    this.who = who;
    this.dueDate = dueDate;
    this.done = false;
}
```

Once we have this new id property, we need to update the code where we create the *Todo* object, to pass in a unique id. But what do we use for that unique id?

OBSERVE:

```
function getFormData() {
    var task = document.getElementById("task").value;
    if (checkInputText(task, "Please enter a task")) return;

    var who = document.getElementById("who").value;
    if (checkInputText(who, "Please enter a person to do the task"))
return;

    var date = document.getElementById("dueDate").value;
    if (checkInputText(date, "Please enter a due date")) return;

    var id = todos.length;
    var todoItem = new Todo(id, task, who, date);
    todos.push(todoItem);
    addTodoToPage(todoItem);
    saveTodoItem(todoItem);
}
```

Each time we create a new Todo object representating a to-do item, we're adding that item to the todos array. Each time we add a new item to the array, the length of the array increases. Because of that, we can use the current length of the array as a unique id. We retrieve the length of the array with todos.length. Once we have an id, which in this case is just a number, we can use that id when we create a new Todo object. Next, we add the new Todo item to the array, and to the page, and then save the new item using our new function `saveTodoItem()`:

OBSERVE:

```
function saveTodoItem(todoItem) {
    if (localStorage) {
        var key = "todo" + todoItem.id;
        var item = JSON.stringify(todoItem);
        localStorage.setItem(key, item);
    }
    else {
        console.log("Error: you don't have localStorage!");
    }
}
```

The `saveTodoItem()` function takes one argument, `todoItem`, which is a *Todo* object with a unique id in the `id` property. The function checks to make sure that the user's browser has a `localStorage` object, and if so, creates a key using the string "todo" concatenated with the `id` of the `todoItem`. So, if we pass in a todo item with an id of 1, then the key will be "todo1".

Now the todoItem is a Todo object, so we have to convert it to JSON before we can store it in Local Storage. We do that using the `JSON.stringify()` method. We now have a key and an item, so we can store the item in Local Storage using `localStorage.setItem()`.

If the user's browser doesn't have `localStorage`, we show an error message in the console. (In a more robust application you might want to have another option.)

Getting To-Do Items from Local Storage

Okay, we've got to-do items stored in Local Storage and they're being displayed on the page as you add them. We put the items in Local Storage so they'll be there when you reload the page (or want to access your items the next day or even a month from now).

We can use `localStorage.getItem()` to retrieve items from Local Storage, but we want to retrieve only items with keys that contain the string "todo." Because we get just the items from Local Storage when we load the page, we need to add all the items in Local Storage to the list you see on the page as well. We already have a function that does that: `addTodosToPage()`. It adds all the items in the `todos` array to the page. So, as we retrieve each item from Local Storage, we'll add it to the `todos` array, then we can call `addTodosToPage()` to add them to the page.

Modify *todo.js* as shown:

CODE TO TYPE:

```
function Todo(id, task, who, dueDate) {
    this.id = id;
    this.task = task;
    this.who = who;
    this.dueDate = dueDate;
    this.done = false;
}

var todos = new Array();

window.onload = init;

function init() {
    var submitButton = document.getElementById("submit");
    submitButton.onclick = getFormData;

    getTodoItems();
}

function getTodoItems() {
    if (localStorage) {
        for (var i = 0; i < localStorage.length; i++) {
            var key = localStorage.key(i);
            if (key.substring(0, 4) == "todo") {
                var item = localStorage.getItem(key);
                var todoItem = JSON.parse(item);
                todos.push(todoItem);
            }
        }
        addTodosToPage();
    }
    else {
        console.log("Error: you don't have localStorage!");
    }
}

~~function parseTodoItems(todoJSON) {~~
    ~~if (todoJSON == null || todoJSON.trim() == "") {~~
        ~~return;~~
    ~~}~~
    ~~var todoArray = JSON.parse(todoJSON);~~
    ~~if (todoArray.length == 0) {~~
        ~~console.log("Error: the to-do list array is empty!");~~
        ~~return;~~
    ~~}~~
    ~~for (var i = 0; i < todoArray.length; i++) {~~
        ~~var todoItem = todoArray[i];~~
~~        todos.push(todoItem);~~
~~    }~~
~~}~~
function addTodosToPage() {
    var ul = document.getElementById("todoList");
    var listFragment = document.createDocumentFragment();
    for (var i = 0; i < todos.length; i++) {
        var todoItem = todos[i];
        var li = createNewTodo(todoItem);
        listFragment.appendChild(li);
    }
    ul.appendChild(listFragment);
}
function addTodoToPage(todoItem) {
    var ul = document.getElementById("todoList");
    var li = createNewTodo(todoItem);
    ul.appendChild(li);
    document.forms[0].reset();
}

function createNewTodo(todoItem) {
    var li = document.createElement("li");
    var spanTodo = document.createElement("span");
    spanTodo.innerHTML =
        todoItem.who + " needs to " + todoItem.task + " by " +
todoItem.dueDate;

    var spanDone = document.createElement("span");
    if (!todoItem.done) {
        spanDone.setAttribute("class", "notDone");
        spanDone.innerHTML = "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
    }
    else {
        spanDone.setAttribute("class", "done");
        spanDone.innerHTML = "&nbsp;&#10004;&nbsp;";
    }

    li.appendChild(spanDone);
    li.appendChild(spanTodo);
    return li;
}

function getFormData() {
    var task = document.getElementById("task").value;
    if (checkInputText(task, "Please enter a task")) return;

    var who = document.getElementById("who").value;
    if (checkInputText(who, "Please enter a person to do the task"))
return;

    var date = document.getElementById("dueDate").value;
    if (checkInputText(date, "Please enter a due date")) return;
    var id = todos.length;
    var todoItem = new Todo(id, task, who, date);
    todos.push(todoItem);
    addTodoToPage(todoItem);
    saveTodoItem(todoItem);
}

function checkInputText(value, msg) {
    if (value == null || value == "") {
        alert(msg);
        return true;
    }
    return false;
}

function saveTodoItem(todoItem) {
    if (localStorage) {
        var key = "todo" + todoItem.id;
        var item = JSON.stringify(todoItem);
        localStorage.setItem(key, item);
    }
    else {
        console.log("Error: you don't have localStorage!");
    }
}
```

Save the changes to *todo.js*, and open or refresh *todo.html* in your browser. You see all the items that you had previously added in the page. You'll be able to add new ones. Give it a try. Remove some items from Local Storage using the browser console tool, and then reload and make sure that what you see in Local Storage matches what you see in the page.

Note You may need to refresh the view of Local Storage in your browser (using the browser tools) or use the JavaScript console to display the `localStorage` object after you add an item to see the updated values.

Let's walk through the changes we made. We add a function `getTodoItems()`, and call that function from the `init()` function, so it runs as soon as the page loads and you see all the items you have stored, as soon as you load the page.

OBSERVE:

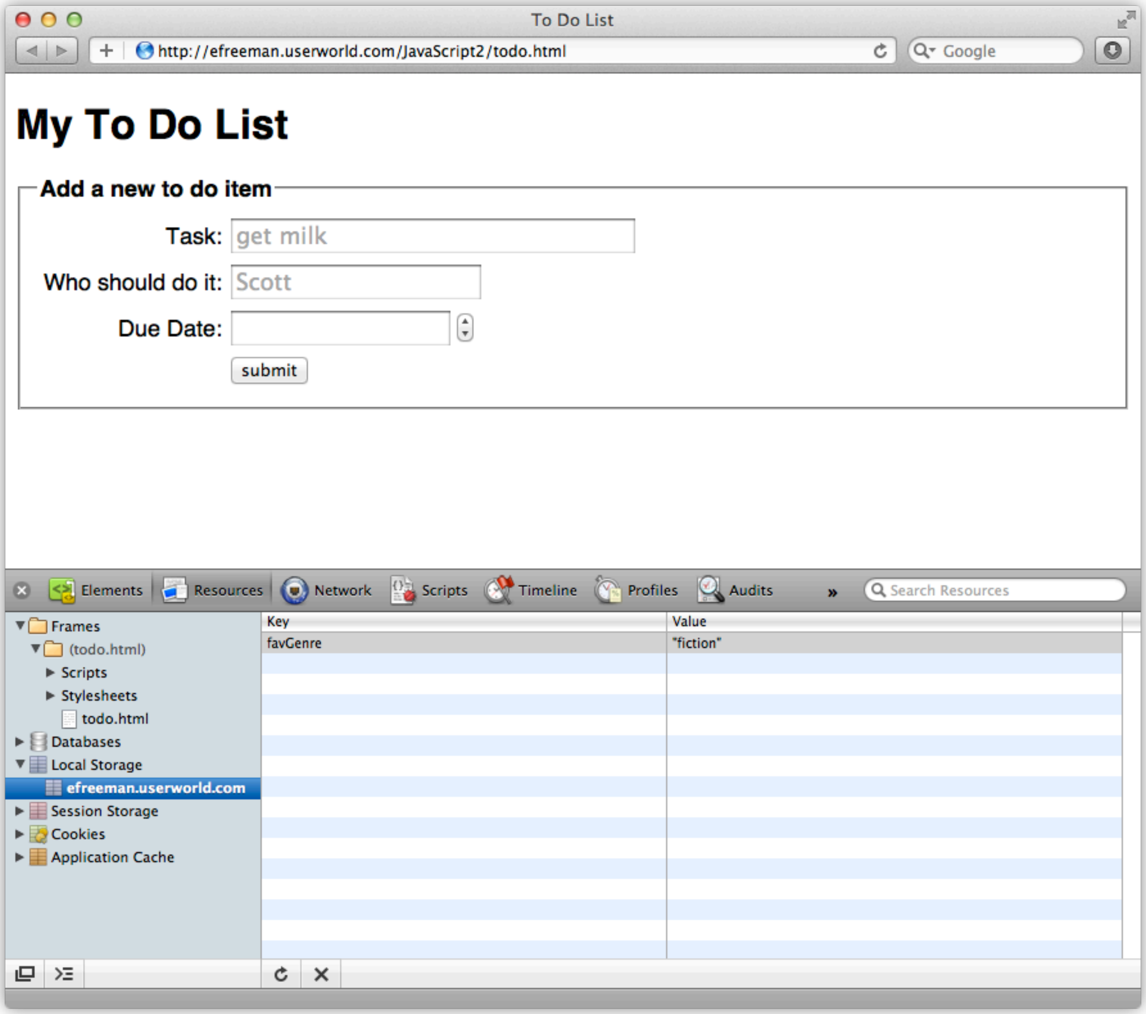
```
function getTodoItems() {
    if (localStorage) {
        for (var i = 0; i < localStorage.length; i++) {
            var key = localStorage.key(i);
            if (key.substring(0, 4) == "todo") {
                var item = localStorage.getItem(key);
                var todoItem = JSON.parse(item);
                todos.push(todoItem);
            }
        }
        addTodosToPage();
    }
    else {
        console.log("Error: you don't have localStorage!");
    }
}
```

In `getTodoItems()`, we check to make sure the `localStorage` object exists— if it doesn't, display a message in the console.

Next, we loop through all the items in Local Storage using the `length` property of the `localStorage` object, but this time, instead of adding every item to the page, we want to add only the items that have the string "todo" in the key. We check to make sure that the key string contains the string "todo", using the String method `substring()` (we'll come back to this method shortly). If it does, then we get the item from Local Storage, and use JSON to `parse()` the string back into a *Todo* object, which we store in the variable `todoItem`. Then we add the `todoItem` to the `todos` array. Finally, after we've added all of the to-do items in Local Storage to the `todos` array, we call the `addTodosToPage()` function, which adds all the to-do items to the page.

We delete the `parseTodoItems()` function from the code. We don't need this function anymore because we parse each item in the loop as we get the items from Local Storage, and we add each item to the array right there. Remember, we used `parseTodoItems()` in the Ajax version of the To-Do List application to parse the JSON we got back from the *XMLHttpRequest*.

If you have no "todo" items in Local Storage, when you load your page, you'll see:



My To Do List

Add a new to do item

Task: get milk

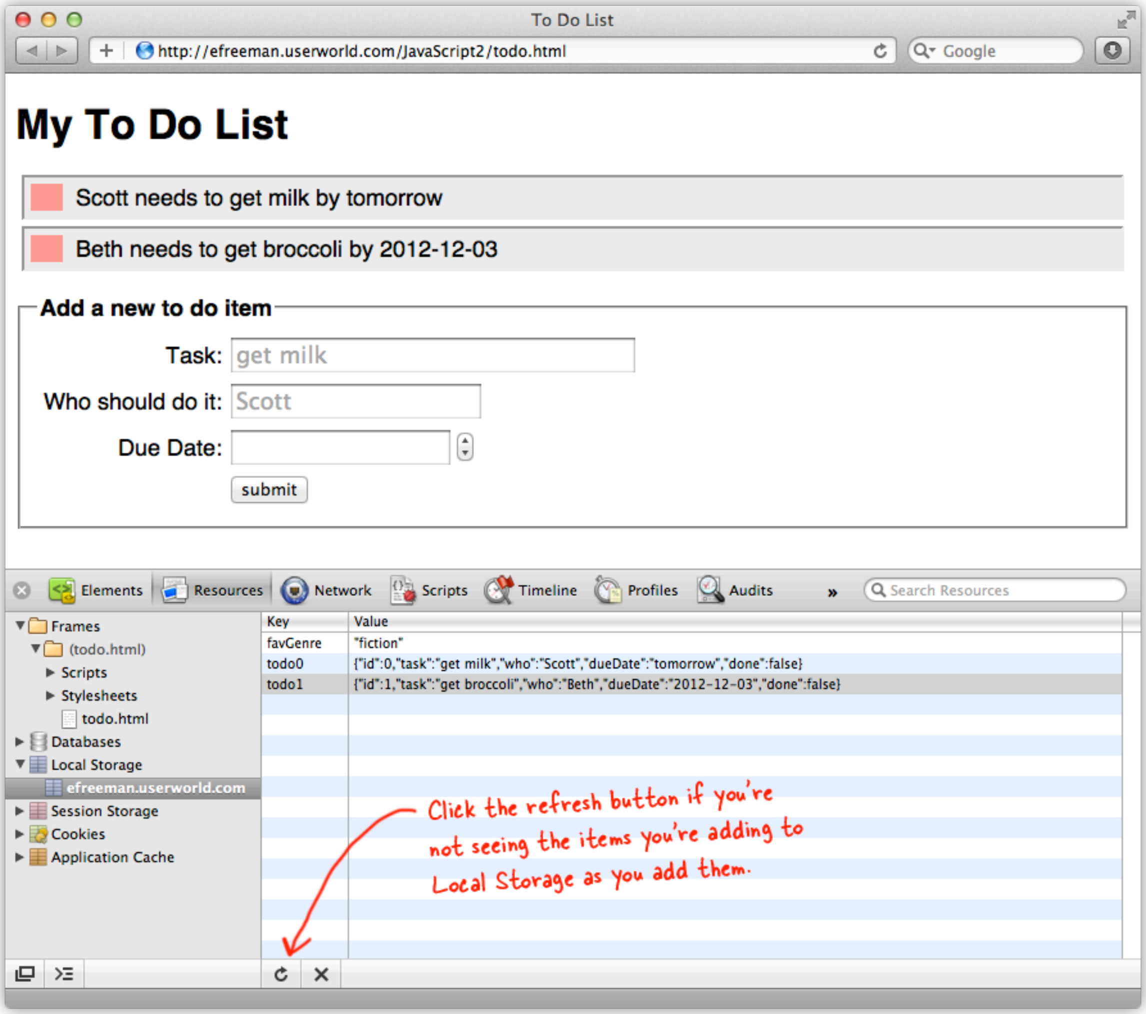
Who should do it: Scott

Due Date:

submit

Key	Value
favGenre	"fiction"

After adding some "todo" items, you'll see:



My To Do List

🟥 Scott needs to get milk by tomorrow

🟥 Beth needs to get broccoli by 2012-12-03

Add a new to do item

Task:	get milk
Who should do it:	Scott
Due Date:	

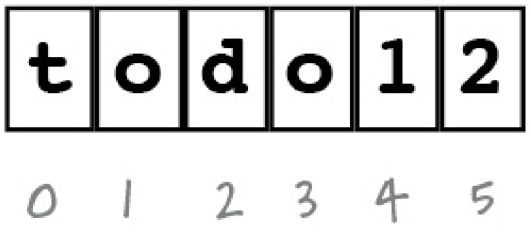
Key	Value
favGenre	"fiction"
todo0	{"id":0,"task":"get milk","who":"Scott","dueDate":"tomorrow","done":false}
todo1	{"id":1,"task":"get broccoli","who":"Beth","dueDate":"2012-12-03","done":false}

Click the refresh button if you're not seeing the items you're adding to Local Storage as you add them.

The String substring() method

In the function we just added, `getTodoItems()`, we used a String method, `substring()`. We haven't talked much about String methods yet. We have a whole lesson devoted to Strings later in this course. For now, let's take a closer look at the `substring()` method to see how it works, and how to use it to pull out only to-do items for the To-Do List application.

Think of a *String* as a set of multiple characters. It's a little bit like an array (but it's definitely NOT an array, so don't confuse the two), in that a character in a string holds a position, like this:



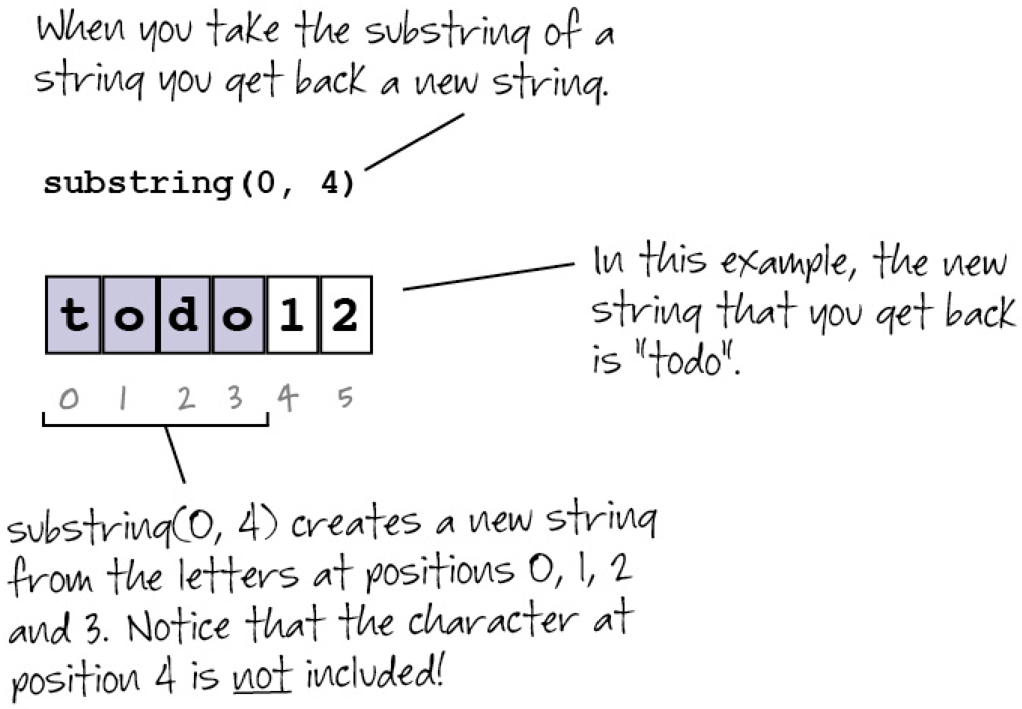
So the string "todo12" has a length of six; six separate characters that together make up the String. We capitalize *String* here because String is a special object in JavaScript. It's not quite like the objects you create in JavaScript, but similar in the sense that a String has properties (such as `length`) and methods (such as `substring()`).

The `substring()` method allows you to take any String and create a new String from it by using a range of characters. The two arguments you pass to the `substring()` method are a *from* value and a *to* value, where *from* is the position of the beginning character you want and *to* is the position of the next character *after* the ending character you want. For example, if you have the string "Hello World!" and you want to get the string "lo w", your code would look like this:

OBSERVE:

```
var str="Hello world!";
var mySubString = str.substring(3,7);
```

The variable `mySubString` will contain "lo w" -- all the characters beginning at position 3 and ending at (but *not including the character at*) position 7. (Remember that string positions start at 0, so "H" is at 0, "e" is at 1, and so on). For our Todo application, we can extract the "todo" portion of the key items using `substring()`, like this:


When you take the substring of a string you get back a new string.
substring(0, 4)
t o d o 1 2
0 1 2 3 4 5
In this example, the new string that you get back is "todo".
substring(0, 4) creates a new string from the letters at positions 0, 1, 2 and 3. Notice that the character at position 4 is not included!

You can use the `substring()` method on any string you create in JavaScript. So, if we have a variable `key` that contains the string "todo12," we can call `substring()` on the `key` variable—`key.substring(0, 4)`—which returns the first four characters of the string (from positions 0, 1, 2, and 3), unless the length of the value in the key is less than four characters. In that case, you get all the letters in the string—so if the `key` contains the string "to," you get "to" as the result.

We can compare the result of calling `substring()` on `key` to a literal string, "todo," to see if they are equal, using the == operator:

OBSERVE:

```
if (key.substring(0, 4) == "todo") {
   ...
}
```

If they are, we know we've found a key for a to-do item. Because the `substring()` method returns a *new* string, we haven't changed the value of `key` at all. In our example, `key` still contains the string "todo12."

In this lesson, we updated the To-Do List application to use Local Storage instead of Ajax. That means the user's to-do items are stored in the browser they used to add items to their to-do list, using the To-Do List application.

We also learned the importance of choosing good keys for your applications that use Local Storage. In this example, we used the string "todo" and a unique id to create a key for to-do list items. It seems to be working well, but can you think of anything that might cause our id / key scheme to fail? What would happen if you deleted a todo item from Local Storage, and then added a new item using the form without reloading the page first? Will the ids always match the position in the todos array when you get all the items from Local Storage, when you first load the page, using the function `getTodoItems()`? If not, could that cause a problem?

Think about those questions in preparation for the next lesson.

More about the material in this lesson

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.