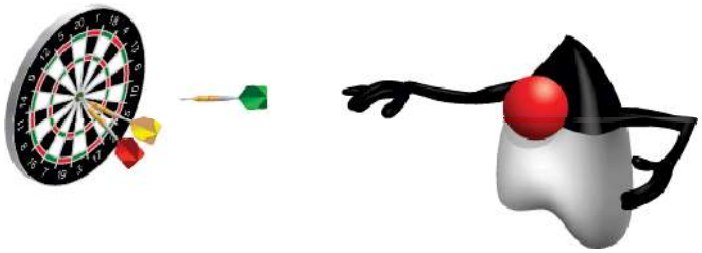
5

The JavaScript API

Objectives

After completing this lesson, you should be able to do the following:

- Validate user input with JavaScript and regular expressions
- Handle multiple values with JavaScript collections
- Manipulate dates with the JavaScript date API



Topics

- Parsing and validating
- URL Encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

Topics

- Parsing and validating
- URL Encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

The JavaScript API

JavaScript contains a set of functions and objects, which you can use to perform a number of tasks, including:

- Converting text to numbers
- Checking types such as NaN and Infinity
- Manipulating text data
- Validating input
- Creating collections of data
- Accessing the browser window, parent document, and so on

Parsing Numbers

Scenario: You want to know how many years the user has left until he is 100 years old.

```
<div>
  Years until you are 100 years old:
  <input type="text" id="ageInput"/>
  <input type="button" value="calculate" id="ageButton"/>
  <div id="ageOutput"> </div>
</div>
```

Adding Events to DOM

```
window.addEventListener("load", function() {…}, false);
```

```
var button = document.getElementById("ageButton");
button.addEventListener("click", function() {…}, false);
```

```
window.addEventListener("load", function() {
  var button = document.getElementById("ageButton");
  button.addEventListener("click", function() {
    // Button click logic goes here
  }, false); // closes the button event handler
}, false); // closes the window event handler
```

Number Parsing and Output

```
var ageOutputElement = document.getElementById("ageOutput");
var ageTextElement = document.getElementById("ageInput");

var ageTextValue = ageTextElement.value;
var age = parseInt(ageTextValue, 10);

var result = "";

if (isNaN(age)) {
  result = "Input a number please.";
  ageTextElement.value = "";
} else {
  result = ((100 - age) + " years before you turn 100!");
}
console.log(result);
ageOutputElement.innerHTML = result;
```

Tip Calculator

Scenario: You and some friends go for lunch and opt to divide the bill, adding a 10% tip.

```
<div>
  Tip Calculator<br>
  <input type="text" id="tipTotalBill"/>
  <input type="text" id="tipNumberOfPeople"/>
  <input type="button" value="calculate" id="tipButton"/>
  <div id="tipOutput"> </div>
</div>
```

Tip Calculator Logic

```
var billTotal = parseFloat(tipTotalBillElem.value);
var people = parseFloat(tipNumberOfPeopleElem.value);
//add the tip
var totalWithTip = billTotal + (billTotal * 0.1);
//divide the total
var dividedTotal = totalWithTip / people;
if (isNaN(dividedTotal)) {
  tipOutputElem.innerHTML = "Please input numbers.";
} else if (isFinite(dividedTotal)) {
  tipOutputElem.innerHTML = "Each of you will pay:"+dividedTotal;
} else {
  tipOutputElem.innerHTML = "Can't divide by 0.";
}
```

`Number.toLocaleString()`

- Use `toLocaleString()` in numbers to get a string representation of a number for display.
- Usage:
- `number.toLocaleString(Locale, Options)`
- Locale is a string representing the BCP 47 language tag (for example, “en-us”).
- `Options` is a JavaScript object that provides specific parameters that are used to display the number.

```
number.toLocaleString("en-US", {
  style: "currency", currency: "USD",
  maximumFractionDigits: 2});
```

Intl.NumberFormat

- To reuse formatting settings, you can use the `Intl.NumberFormat` object:

```
var numberFormat = new Intl.NumberFormat("en-US", { style: "currency",
    currency: "USD",
    maximumFractionDigits: 2});
var string1 = numberFormat.format(number1);
var string2 = numberFormat.format(number2);
```

HTML5 Number Input and JavaScript

- As a safety measure and to ensure compatibility, always parse values by using JavaScript.

```
<input type="number" id="numberInput">
```

- Parse by using:

```
var elem = document.getElementById("numberInput");
var number = parseInt(elem.value);
```

Topics

- Parsing and Validating
- URL Encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

String Creation

- `var s = "this is a string"`
- `var s = new String("this is a string")`

- Escape sequences: `\'` `\"` `\\` … and so on

- What about international characters?

  `var s = "中文 español English हिन्दी العربية português বাংলা русский 日本語 ਪੰਜਾਬੀ 한국어"`
- Use Unicode:

  `var s = "\u4E2D" = "中"`

String Manipulation

One of the most common operations that you perform is string manipulation:

- Concatenation: `"hello "+"world" = "hello world";`
- CharAt: `"hello world".charAt(4) = 'o'`
- indexOf: `"hello world".indexOf("world") = 6`
  `"hello world".indexOf("no") = -1`
- replace: `"hello world".replace("world", "John") = "hello John"`
- split: `"hello world".split(" ") = ["hello","world"]`

String Manipulation

- substring: `"hello world".substr(3) = "lo world"`
- toLowerCase: `"Hello World".toLowerCase() = "hello world"`
- toUpperCase: `"Hello World".toUpperCase() = "HELLO WORLD"`
- trim: `"    Hello World    ".trim() = "Hello World"`

Validating and Parsing String Values

Scenario: Data is provided to you in comma-separated values and you must display it in your application.

```
var data = "John,Doe,32,1982,10,23,153.25,  A  ,";
```

Splitting the String

Splitting the text creates an array of strings:

```
var data = "John,Doe,32,1982,10,23,153.25,  A  ,";

var splitData = data.split(",");

console.log(splitData[0]); // "John"
console.log(splitData[1]); // "Doe"
console.log(splitData[2]); // "32"
console.log(splitData[3]); // "1982"
console.log(splitData[4]); // "10"
console.log(splitData[5]); // "23"
console.log(splitData[6]); // "153.25"
console.log(splitData[7]); // "  A  "
```

Topics

- Parsing and validating
- URL encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

JavaScript Dates

JavaScript provides a Date object for handling dates:

```
new Date();
// Creates a new date with the current date and time.

new Date(value);
// Value must be a numeric integer value representing the
// number of milliseconds since January 1st 1970 00:00:00 UTC.

new Date(dateString);
// dateString must be a string in IETF-compliant RFC 2822
// timestamp or ISO8601 format.

new Date(year, month, day, hour, minute, second,millisecond);
// A set of integers representing each value for the date.
```

Date String Examples

You can create dates with the following string formats:

```
// RFC 2822
"Aug 9, 1985"
"Wed, 09 Aug 2014 00:00:00 GMT"
"Wed, 09 Aug 1995 00:00:00"
"01 Jan 2011 00:00:00 GMT-0400"

// ISO 8601

"2011-05-22"
"2011-05-22T14:48:00"
```

Date Object

- Create Date objects to manipulate dates.
- Using dates, you have methods to get and set specific fields in a date.

```
Date date = new Date("2011-05-22T14:48:00")
Console.log(date.getDate());
// It will return 22 which is the day of the month.
```

Modifying a Date

- Use `Date.setTime(timestamp);`.
- The time stamp is the number of milliseconds since January 1st 1970 00:00:00 UTC.

- Alternatively, you can use the set methods in a date:

```
date.setDate(14);
// sets the day of the month to 14. All other
// fields (month, year, etc)remains unchanged.
```

Date Operations

Given:

```
var startDate = new Date();
//Some complex code..
var endDate = new Date();
```

- To calculate the elapsed time between two dates in milliseconds:

```
var elapsed = endDate - startDate;
```

- Is the same as:

```
var elapsed = endDate.getTime() - startDate.getTime();
```

Reading and Parsing Dates

Scenario: A user inputs his or her birth date and you want to tell the user how many hours are left before his or her next birthday.

```
var nextBirthDay = new Date();
nextBirthDay.setDate(parseInt(dayElement.value));
nextBirthDay.setMonth(parseInt(monthElement.value)-1);
nextBirthDay.setHours(0);
nextBirthDay.setMinutes(0);
if (nextBirthDay.getTime() < Date.now()) {
 //It is on next year.
 nextBirthDay.setFullYear(nextBirthDay.getFullYear()+1);
}
var milliseconds = nextBirthDay.getTime() - Date.now();
var hours = Math.round(milliseconds / (1000 * 60 * 60));
var days = Math.round(hours / 24);
var result = hours + " hours (" + days + " days) left
before birthday!";
```

Solution: JavaScript String Manipulation

```
var data = "John,Doe,32,1982,10,23,153.25,  A  ,";
var person = {};
var splitData = data.split(",");
person.firstName = splitData[0];
person.lastName = splitData[1];
person.age = parseInt(splitData[2]);
person.credit = parseFloat(splitData[6]);
person.rating = splitData[7].trim();
var year = parseInt(splitData[3]);
var month = parseInt(splitData[4])-1;
var day = parseInt(splitData[5]);
person.birthdate = new Date(year, month, day, 0, 0, 0, 0);
```

Topics

- Parsing and validating
- URL encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

Regular Expressions

- You can use regular expressions to search for patterns on strings.

- Regular expressions are independent JavaScript objects:

```
var regex = new RegExp(pattern [, flags]);
```

- Alternatively, you can use the Regex literal:

```
var regex = /pattern/flags
```

Sample Regular Expression

```
var regexp = /oracle/i
```

Pattern Flags

Matches:

```
"Visit oracle.com for more information"

"ORACLE PRODUCTS"

"Oracle provides courses in Oracle University"
```

Sample Regular Expression

```
var regexp = /[a-z]*/
```

Pattern: All lowercase letters

Matches:

```
"Visit oracle.com for more information"

"ORACLE PRODUCTS"

"Oracle provides courses in Oracle University"
```

Submatched Parentheses Expressions

```
var regexp = /(o+)r/i
```

Pattern: One or more O's followed by an r
Submatch: One or more O's at the beginning of the pattern

Will match:

```
"Visit oracle.com for more information"

"ORACLE PRODUCTS"

"Oracle provides courses in Oracle University"
```

Regular Expression Methods

- `exec([String]):Object`
  Executes a search in the string for the pattern, returns an object with information about the search, and updates the `regexp` object

- `test([String]):Boolean`
  Returns `true` if the string contains one or more matches of the search pattern

String RegExp Methods

`regexp` objects can be used in the following string methods:

- `String.replace(regexp, newString)`
  Replaces all the matched instances of the pattern with the new string
- `String.match(regexp)`
  Returns an array of strings with all the matched patterns

RegExp Example

Scenario: You want to validate the email address provided by the user.

A simple pattern to match emails is:

```
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}
```

Matches email names | @ | Host names | dot | 2 to 4 characters com, co, jp

Topics

- Parsing and validating
- URL encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

JavaScript Arrays as Collections

To modify arrays and treat them as collections, you can use several methods:

- `array.pop()` removes and returns the last element of the array.
- `array.push(value)` adds the value at the end of the array.
- `array.shift()` removes and returns the first element of the array.
- `array.unshift(value)` adds the value at the start of the array.
- `array.splice()` removes and replaces items.
- `array.indexOf(value)` gets the index of value.
- `array.join()` joins all items in a single string.

Array Methods

- pop:

```
var shapes = ["triangle", "circle", "square"];
var shape = shapes.pop();
shape;                  // "square"
shapes;                 // ["triangle", "circle"]
```

- push:

```
var months = ["Jan", "Feb", "Mar"];
months.push("Apr","May","Jun");
months;   // ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]
```

Array Methods

- shift:

```
var shapes = ["triangle", "circle", "square"];
var shape = shapes.shift();
shape;                  // "triangle"
shapes;                 // ["circle", "square"]
```

- unshift:

```
var elements = ["fire", "water", "earth", "air"];
elements.unshift("fifth element");
elements;     // ["fifth element", "fire", "water",
                  "earth", "air"]
```

Array.Splice()

- Splice is used to add and remove elements at a certain index.

```
array.splice(index, removeCount, element1,
element2, … elementN);
```

- Splice removes the `removeCount` elements from the array at the specified index, and then adds all the other elements in place of the removed ones (if any).

Array.join()

- Use `Array.join` to create a string representation of the array contents.

```
[1,2,3,4].join(", "); // "1, 2, 3, 4"

["a","b","c"].join("-"); // "a-b-c"
```

Array.indexOf()

- indexOf:

```
var items = ["pencil", "scissors", "eraser", "tape"];
var index = items.indexOf("eraser");
index;                    // 2
index = items.indexOf("glue");
index;                    // -1
```

Reading and Parsing HTML Lists with JS Arrays

Scenario: The user selects elements from a list and you want to display the elements that the user selected.

```
var listElement =
document.getElementById("listElement");
var result = [];
for(var i = 0; i<listElement.length;i++){
  if(listElement[i].selected){
    result.push(listElement[i].value);
  }
}
console.log(result.join(", "));
```

Solution: JavaScript Arrays

```
//Add the following:
animals.pop();
animals.pop();
numbers.push(10);
floats.splice(2,1);
colors.splice(3,1,"orange");
people=[];
// people.splice(0,people.length);
// people.length=0;
// while(people.length>0){people.pop();}
functions.push(function(){return true;});
```

Topics

- Parsing and validating
- URL encoding
- Strings
- Dates
- Regular expression object
- JavaScript arrays and collections
- Objects

Objects creation (refresher)

Inline direct instance (singleton):

```
var object = {
  property1:value,
  method:function(){},
  property2: value};
```

or

```
var object = new
function(){
  this.property1=value1;
  this.method=function(){};
  this.property2=value2;
}
```

Class function declaration:

```
function className(){
  this.property1=value1;
  this.method=function(){};
  this.property2=value2;
}

var object = new className();

// className.prototype
```

Static Object Methods

- Some static methods of the `Object` constructor:
  - `Object.create()`
  - `Object.defineProperty()`
  - `Object.getOwnPropertyDescriptor()`
  - `Object.getOwnPropertyNames()`
  - `Object.getPrototypeOf()`

```
var obj = { a : 45 };
Object.defineProperty(obj, "double_a", {get :
    function(){ return this.a * 2; }});
Object.defineProperty(obj, "modify_a", {set :
    function(x){ this.a -= x; }});
```

Prototypes

- All JavaScript objects have a prototype property.
- Methods and “static” values are usually declared in the prototype.

- A class function declaration by using a prototype is as follows:

```
function className(){
  this.property1=value1;
  this.property2=value2;
}

className.prototype.method = function(){}

var object = new className();
```

Object Methods

- Object methods inherited from Object.`prototype`:
  - `isPrototypeOf(obj)`
  - `toLocaleString()`
  - `toString()`
  - `valueOf()`

`ToString` Object Method

Overwrite the `toString()` method to provide an easy way to represent objects as strings:

```
function Person(_name, _lastName) {
  this.name = _name;
  this.lastName = _lastName;
}

Person.prototype.toString = function() {
  return this.lastName + ", " + this.name;
};

var person = new Person("John","Doe");
console.log(person.toString()); // "Doe, John"
```

Where Can I Learn More?

Resource	Website
JavaScript Standard built-in objects	https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects
ECMA Script	http://www.ecmascript.org/
Regular Expressions	http://www.regular-expressions.info/
Regex Tester	http://regexpal.com/

Summary

In this lesson, you should have learned how to:

- Parse and validate user input from input elements by using JavaScript
- Parse and manipulate dates
- Store multiple values and objects inside JavaScript arrays and use them as collections
- Add and modify existing object methods by using prototypes