Functions

Chapter 3
Objectives

• Learn how to use functions to organize your JavaScript code
• Learn about applying functions in the context of objects
Working with Functions

• Procedures similar to the methods associated with an object
  – e.g. `document.write()`

• Make it possible to treat a related group of JavaScript statements as a single unit

• Especially useful if you are using the same code multiple times

• Must be contained within a `<script>` element
Defining Functions

```javascript
function nameOfFunction(parameters) {
  statements;
}
```

- **Parameter**: variable that is used locally within a function, in brackets following a `function name`
- **Arguments** is another word for `parameters`
- Multiple parameters separated by commas
- Note: parameter list can be empty
Example

function displayMyInfo(name, age)
{
    document.write("<p>My name is:" + name + "</p>"肚
    document.write("<p>My age is:" + age + "</p>"肚
}

Calling Functions

• Function definition does not automatically execute
• Need to invoke or *call* a function to execute it
• Function call: function name followed by (round) brackets with required variables or values
• These values correspond the arguments or parameters
Example

displayMyInfo("John",30);
Calling Functions...

• It is good practice to place:
  – Function definitions within the head
  – Function calls within the body

• Attempts to call a function before it has been created receive an error

• A JavaScript program is composed of all the <script> sections within a document
Functions results

• Functions can also return a value
• `return` statement delivers a value to the statement that called the function

```javascript
function averageNumbers(a, b, c) {
    var sum_of_numbers = a + b + c;
    var result = sum_of_numbers / 3;
    return result;
}

var returnValue = averageNumbers(1,2,3);
```
Optional arguments

• All function arguments are optional
For example, we can call the previous function as:

```javascript
var returnValue = averageNumbers(1);
```

In this case, variables `b` and `c` will be set to `undefined`
Scope of Variables

- Where in the program can a variable be used?
  - **Global variable**: is declared outside a function and is available to all parts of your program
    - global variables defined without the `var` keyword
  - **Local variable**: is declared inside a function and is only available within that function
    - declared using `var`

- If a global and a local variable have the same name, the local variable takes precedence when its function is called
JavaScript Built-in Functions

- Some of the built-in functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decodeURI(string)</td>
<td>Decodes text strings encoded with encodeURI()</td>
</tr>
<tr>
<td>decodeURICOMPONENT(string)</td>
<td>Decodes text strings encoded with encodeURICOMPONENT()</td>
</tr>
<tr>
<td>encodeURI(string)</td>
<td>Encodes a text string so that it becomes a valid URI</td>
</tr>
<tr>
<td>encodeURICOMPONENT(string)</td>
<td>Encodes a text string so that it becomes a valid URI component</td>
</tr>
<tr>
<td>eval(string)</td>
<td>Evaluates expressions contained within strings</td>
</tr>
<tr>
<td>isFinite(number)</td>
<td>Determines whether a number is finite</td>
</tr>
<tr>
<td>isNaN(number)</td>
<td>Determines whether a value is the special value NaN (Not a Number)</td>
</tr>
<tr>
<td>parseFloat(string)</td>
<td>Converts string literals to floating-point numbers</td>
</tr>
<tr>
<td>parseInt(string)</td>
<td>Converts string literals to integers</td>
</tr>
</tbody>
</table>

Table 3-1  Built-in JavaScript functions
Built-in Functions - Examples

• Using values from a form field text box (string)

• Check if a string is a number: isNaN(string)
  isNaN("34352") returns false
  isNaN("34-352") returns true

• eval(string)
  return $\text{ReturnValue} = \text{eval}("2+4")$;
Built-in Functions - Examples

• `parseInt(string)`

```javascript
var x=2, y="4";
var result=x+y;
document.write("<p>"+result+"</p>" consultancy);
result=x+parseInt(y);
document.write("<p>"+result+"</p>"");
```
Summary

• Functions are similar to the methods associated with an object
• Variable scope refers to where a declared variable can be used

Further reading:
• Chapters 3
Creating own Methods

- Methods are functions associated with an object
- Adding a method is similar to adding properties
- Use keyword `function` followed by round brackets

```javascript
object.functionName = function (arguments) {
    statement(s)
}
```

- Within function, use `this` to refer to the current object and to access its properties
Examples

myObject.toString = function ()
{
    return this.firstName" "+this.lastName;
};
Adding pre-defined functions

• Can also add existing functions as member to an object

```javascript
function aFunction() {
    statements();
}

myObject.memberFunction = aFunction;
```

• Note: omit argument brackets
Advanced JavaScript

• There are no classes as such in JavaScript
• However, custom-made objects can be created using a constructor
• Constructors are also functions, and the convention is to start constructor functions with a capital letter (note: this is not necessary!)
function Person(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
    this.toString = function() {
        return firstname + " " + lastname;
    }
}

var amy = new Person("Amy","Winehouse");