

Chapter 10

How to create and use functions

How to create an anonymous function

```
var display_error = function ( message ) {  
    alert("Error: " + message);  
}
```

How to create a named function

```
function isEven (value) {  
    return value % 2 == 0;  
}
```

A function with no parameters

```
var coin_toss = function() {  
    return ( Math.random() > 0.5 ) ? "Heads" : "Tails";  
}
```

A function with three parameters

```
var avg_of_3 = function( x, y, z ) {  
    return ( x + y + z ) / 3;  
}
```

Calling a function that returns a value

```
var average = avg_of_3( 5, 2, 8 );    // average is 5  
alert( coin_toss() );                // "Heads" or "Tails"
```

Calling a function that doesn't return a value

```
display_error("Value out of range");
```

Terms

- An *anonymous function* isn't given a name in its function declaration.
- A *named function* is given a name in its function declaration.
- When you code a function, you code a list of *parameters* that are accepted by the function.
- When you *call* a function, you code a list of *arguments* that will be passed to the function.

A function that accepts a primitive type (a string)

```
var add_timestamp = function ( text ) {  
    var now = new Date();  
    text = text + " " + now.toString();  
    alert(text);  
}
```

Code that passes an argument to the function

```
var message = "Error: Value out of range.";  
add_timestamp(message); // displays message and timestamp
```

Code that displays the argument

```
alert(message); // message hasn't been changed
```

A function that accepts an object type (an array)

```
var uppercase_first = function ( x ) {  
    x[0] = x[0].toUpperCase();  
}
```

Code that passes an argument to the function

```
var fruits = [ "apple", "orange" ];  
uppercase_first(fruits);
```

Code that displays the argument

```
alert( fruits[0] );          // fruits has been changed!
```

Terms

- The *primitive types* are numbers, strings, and Booleans.
- When a primitive type is passed to a function, it is *passed by value*.
- When an object is passed to a function, it is *passed by reference*.

A function that uses local and global scope

```
var display_message = function() {  
    if ( message == undefined ) {  
        var message1 = "Unknown message.";  
        alert ( message1 );           // local scope  
    } else {  
        alert(message);               // global scope  
    }  
}
```

Global var is available to all functions

```
display_message();           // Displays "Unknown message."  
var message = "Test.";  
display_message();           // Displays "Test."
```

Local var is only available within the function

```
alert ( message1 );         // Causes runtime error
```

Named functions are created before execution

```
alert( coin_toss() );    // Displays Heads or Tails

function coin_toss() {
    return (Math.random() > 0.5) ? "Heads" : "Tails";
}
```

Anonymous functions are created in sequence

```
alert( coin_toss() );    // Causes a runtime error

var coin_toss = function () {
    return (Math.random() > 0.5) ? "Heads" : "Tails";
}
```

Terms

- The *scope* of a variable or function determines what code has access to it.
- JavaScript uses *lexical scope*.
- Any variable created outside of a function has *global scope* and is available to all functions.
- Any variable created inside a function has *local scope* and is only available within that function.

A function that uses the arguments property

```
var isEven = function (value) {  
    return arguments[0] % 2 == 0;  
}
```

How to determine the number of arguments

```
var count_args = function () {  
    alert("Number: " + arguments.length );  
}
```

```
count_args( 1, "Text", true );    // Displays "Number: 3"
```

A function that handles fewer arguments

```
var pad_left = function(text, width, pad) {  
    if ( arguments.length < 2 ) return "";  
    if ( arguments.length == 2 ) pad = " ";  
    while( text.length < width ) {  
        text = pad + text;  
    }  
    return text;  
}
```

```
alert( "Welcome to " + pad_left("JavaScript", 15) );  
// Displays "Welcome to      JavaScript"
```

A function that handles more arguments

```
var average = function () {  
    if ( arguments.length == 0 ) return 0;  
    var sum = 0;  
    for ( var i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum / arguments.length;  
}  
  
alert ( average(8, 15, 5, 10) ); // Displays 9.5
```

A recursive function that searches an array

```
var search = function(needle, haystack, lo, hi) {
  if ( arguments.length == 2 ) {
    lo = 0;
    hi = haystack.length - 1;
  }

  var middle = Math.ceil( (hi + lo) / 2 );

  if ( hi < lo ) return -1;
  if ( hi == lo ) {
    if ( needle == haystack[middle] ) {
      return middle;
    } else {
      return -1;
    }
  }
  if ( needle == haystack[middle] ) {
    return middle;
  } else if ( needle < haystack[middle] ) {
    return search( needle, haystack, lo, middle -
  } else if ( needle > haystack[middle] ) {
    return search
  }
}
```

Code that creates a sorted array of 256 numbers

```
var numbers = [];  
for ( var i = 0; i < 256; i++) {  
    numbers[i] = random_number(1,1000);  
}  
  
var numeric_order = function (a,b) {  
    if ( a < b ) return -1;  
    if ( a > b ) return 1;  
    return 0;  
}  
numbers.sort(numeric_order);
```

Code that uses the search function on the array

```
var number, position;
do {
    number =
        prompt("Number to find (click cancel to quit):");

    if ( number == null ) break;

    number = parseInt(number);
    if ( isNaN(number) ) continue;

    position = search(number, numbers);
    if ( position == -1 ) {
        alert(number + " is not in the list.");
    } else {
        alert(number + " is at position " + position + ".");
    }
} while ( true );
```