JavaScript

Notes for programming in JavaScript

Methods

Method	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array passes a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that passes a test
find()	Returns the value of the first element in an array that passes a test
findIndex()	Returns the index of the first element in an array that passes a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Checks if an array contains the specified element
indexOf()	Searches an array for an element and returns its index
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Searches an array for an element, starting at the end, and returns its position
map()	Creates a new array by calling a function for each array element
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

Functions

A variable declared inside a function is only visible inside that function.

A function can access a variable declared outside of the function, as long as that variable is not inside another function.

A function can also modify a variable declared outside of the function, as long as the variable is not inside another function.

If a same-named variable is declared inside the function then it shadows the outer one. For instance, in the code below the function uses the local userName. The outer one is ignored:

```
let userName = 'John';

function showMessage() {
  let userName = "Bob"; // declare a local variable

  let message = 'Hello, ' + userName; // Bob
  alert(message);
}

// the function will create and use its own userName
showMessage();

alert( userName ); // John, unchanged, the function did not access the outer
variable
```

Global Variables

Variables declared outside of any function, such as the outer userName in the code above, are called global. Global variables are visible from any function (unless shadowed by local variables). It's good practice to minimize the use of global variables. Modern code has few or no globals. Most variables reside in their functions. However, sometimes they can be useful to store project-level data.

Parameters / Arguments

A function receives parameters, also known as arguments. The parameters are implicitly converted to local variables within the function. When a function is defined with parameters that are not included when the function is called, the parameters become "undefined" variables within the function.
If a parameter has a default value the default value can be defined in the function:

```
function getCountryCode(country = "United States") {
  // if the function is called with no parameter, the default is used
}
```

The default parameter may be another function, instead of a value, as in this example:

```
function getUserInfo(userid = runTestInstead()) {
  // the function runTestInstead will be called if no userid is included
}
```

The return statement

The return statement within a function returns a value, an array, an object or another function, and then terminates the function.

Switch statement

```
switch (action) {
  case 'draw':
    drawIt();
    break;
  case 'eat':
    eatIt();
    break;
  default:
    doNothing();
}
```

```
action = 'something else';

switch(action) {
  case 'drink':
    console.log(action + ' water');
    break;
  case 'eat':
    console.log(action + ' food');
    break;
  default:
    console.log('nothing to see here');
}
```

Objects

There are two basic ways to create an empty object:

```
var obj = new Object();
var obj = {};
```

Ternary operator

```
var test = {prop1: 'this is prop 1', prop2: 'this is prop 2'};

if (test.prop1 == 'this is prop 1') {
  console.log('yes');
} else {
  console.log('no');
}

test.prop1 = 'this is prop 1' ? console.log('yes') : console.log('no');
```

The rest operator

```
function sum(...arg)
```

This means the function can accept a varying number of arguments, and JavaScript will put the arguments into an array. The rest operator must be the last parameter passed to a function. That's why it's called "the rest."

Default parameters

Functions can be given default parameters:

```
function interest(principal, rate, years = 5)
```

If you set a default value for one parameter, all the parameters that follow must also have default values, so if you want only one parameter to have a default value, put it at the end.

Hoisting

Functions declarations can be called before they are declared. The functions are "hoisted" to the top in the run engine. Function expressions are immediately invoked. "var" is hoisted, but "let" and "const" are not.

Hoisting class declaration vs. class expression

[x]

Factory Functions vs. Constructor Functions

[x]

Classical Inheritance vs. Prototypical Inheritance

[x]

Instance methods vs. static methods

[x]