servicenow

ServiceNow JavaScript Primer

Contents

Overview

JavaScript is OOP (Object Oriented Programming) – this means you build smaller objects that make up the whole. There are 3 "flavors" of JavaScript:

1. Client-Side JavaScript (CSJS) – an extended version of JavaScript that enables the enhancement and manipulation of web pages and client browsers

2. Server-Side JavaScript (SSJS) – an extended version of JavaScript that enables back-end access to tables, databases, file systems, and servers (like the aforementioned ServiceNow GlideRecords)

3. Core JavaScript – the base JavaScript language used by both client and server-side JavaScripts.

Core JavaScript encompasses all of the statements, operators, objects, and functions that make up the basic JavaScript language. JavaScript is the world's most popular programming language, used on more platforms and in more languages than any other programming language in history. It's been licensed by more than 175 companies for inclusion in their web tools. JavaScript is even available as a standalone scripting language.

As you can see, core JavaScript contains objects that are applicable to both client and server. If you know core JavaScript, you can easily write client-side and server-side JavaScript. Again, the only distinction is that client-side and server-side JavaScript have additional objects and functions that you can use that are specific to client-side or server-side functionality. Any JavaScript libraries (.js files) you create in core JavaScript can be used on both the client and the server with no changes whatsoever.

When writing JavaScript, you have to follow strict syntax rules. While white space means nothing when you are coding in, for example, HTML, how much space you leave between words or paragraphs doesn't matter – the “shape” of your HTML code doesn’t matter.

The opposite is true of JavaScript – it does have a shape. So, if you open a text editor and type:

```
<script language="JavaScript" type="text/javascript">
<!--
  document.write("<font color='red'>This Is Red Text</font>")
// -->
</script>
```

Stick it in the middle of some HTML code and you get: This is Red Text

document.write("This Is Red Text")

So, in this example JavaScript, the **document**, in this case the HTML document, is announced. The document will be altered by **write**-ing something to it. What will be written to the document is inside the parentheses **("This Is Red Text")**

Remember, JavaScript is an object-oriented language – this means functions and properties are grouped into logical units called **objects**.

If you want to write to the status bar at the bottom of the browser window, you'd set the status property of the Window object like this:

window.status =(“God Save the Queen”)

Every element of a Web page gets represented as an object. The objects are related to one another in a hierarchical structure.

The Language

Syntax

Basic need-to-know:

- EVERYTHING is case-sensitive

```
var stringA = "String A";
var stringa = "String a";
alert(stringA == stringa); // false
```

- Use as much, or as little extra whitespace as you like.

```
var stringA =           "String A";
var string="String B";
```

- Semicolons are JavaScript's equivalent of a period. After you complete a statement (sentence), end it with a ";" character.
- When defining string values ("this is a string of characters"), single (') and double (") quotes are interchangeable, as long as the closing matches the opening.
- Parentheses are used for 2 reasons:

1. Checking equality (evaluating 2 or more items to simply "true" or 'false')

```
if (stringA == stringa){
   // do something
}
```

2. Sending or receiving a value

```
var product = multiply(7, 3); // send 7 and 3
function multiply(a, b){return a*b;} // receive 7 as "a" & 3 as "b"
```

- Comments are a way to explain to others (or leave reminders for yourself) what you're doing, and are denoted one of two ways:

1. Single- line comments are denoted by a double slash "//"

```
// This is a friendly reminder
var product = multiply (4, 6);
```

2. Multi-line comments are denoted by a slash-star to start "/*" and a star-slash "*/" to end.

```
/*
// This is a friendly reminder that the following lines
// do not get evaluated
var product = multiply(8, 7); product = multiply(product, 7);
*/
```

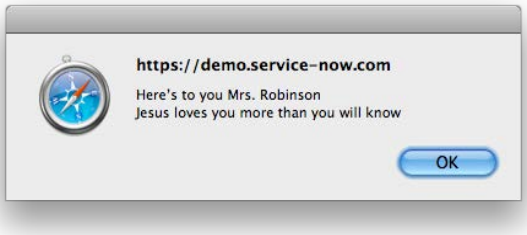
- When working with strings (anything enclosed by quotation marks), there are some characters that have special meaning that left alone will break your script. These include apostrophes (') and double quotes ("). For these, you need to use an "escape character". An escape character enables you to output characters you wouldn't normally be able to, usually because they will be interpreted differently than what you intended.

  There are also a few special characters (or escape sequences) that represent those characters that cannot be typed from the keyboard. For example, '\n' means start a new line.

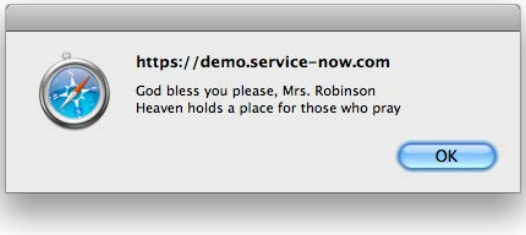
  Here's an example of it being used two different ways. The '\n' inserts a new line. The line break I've added for readability in stringA between the two lines of the song (along with a "+" to add or "concatenate" the two strings together), doesn't actually add a line break in the browser alert—the "\n" does.

```
var stringA = 'Here\'s to you Mrs. Robinson\n' +
              'Jesus loves you more than you will know';
var stringB = 'God bless you please, Mrs. Robinson\nHeaven holds a place for those who pray';
alert(stringA);
alert(stringB);
```

String A:



String B:



Also note that there are a number of "reserved" words that have special meaning in JavaScript. You will see some of these in action later.

They are:

break	case	catch	Continue	default	delete	do
else	false	finally	for	function	if	in
instanceof	new	null	return	switch	this	throw
true	try	typeof	var	void	while	with

Variables and Data Types

Variables

Variables are the workhorse of JavaScript. They allow you to create a container that holds a piece of data, then reference and manipulate what it contains. One thing to remember about JavaScript is that variable can contain *any* type of data—string values, integers, objects, arrays, functions.

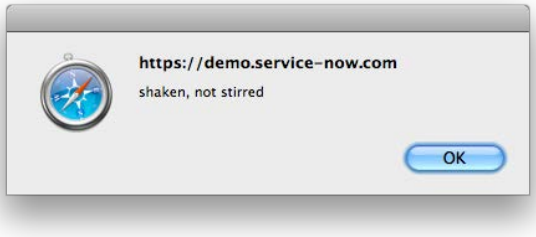
You "open" or initiate a new variable by using the "var" keyword:

```
var myVariableName = "My Variable Value";
```

Data Types

There are five basic data types in JavaScript:

1. **Strings:**
   ```
   var string = "This is a string of characters";
   ```
2. **Numbers:**
   ```
   var number = 127;
   ```
3. **Boolean:** The equivalent of a 1 or 0 (on/off, true/false)
   ```
   var bool = true;
   ```
4. **Array:** This holds a number of values (not all the values have to match data types, be careful)
   ```
   var array = ["value1", "two", 2, "value4", false];
   ```
5. **Object:** Objects area way to hold structured data – like an array, but with “named” properties
   ```
   var object = {name: "My first object",
                 type: "Object-literal",
                 color: "purple",
                 contents: "shaken, not stirred"};
   alert(object.contents); // alerts "shaken, not stirred" in the browser
   ```



Functions

Functions serve 2 major purposes:

1. Make reusable code (modularization)
2. Clean your code for legibility

Defining a Function

We have the following parts:

1. **Name:** Just like a variable where we call "var" and give that a name, to create a function, we call "function" and give it a name in the same way

   ```
   function myName( ) {   }
   ```

2. **Arguments:** These are the *inputs* that the function will manipulate – separate multiple inputs with commas

   ```
   function myName(a, b) {   }
   ```

3. A "**block**" that gets executed. A block is nothing more than a "paragraph" – one or more statements grouped and executed or "read" in sequence

   ```
   function myName(a, b) {
   var c = a*b;
   return c;
   }
   ```

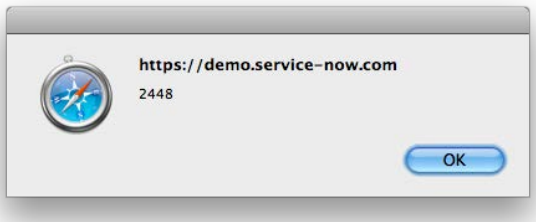
Modularization

This practice allows you to define once, then use many times. Take a look at our multiplication function:

```
function multiply(a, b){
    var c = a*b;
    return c;
}
```

Once it's defined, we can use it as many times as we like:

```
var prod1 = multiply(3, 4);
var prod2 = multiply(prod1, 17);
var prod3 = multiply(prod2, prod1);
alert(prod3); //result is 2448
```



Readability

As of now we haven't seen code that is long enough to be difficult to read or follow, but when your scripts get longer, you'll run into the readability problem.

Here's an example of a longer piece of code (in ServiceNow, this is the out-of-box VIP caller highlighting script):

```
function onChange(control, oldValue, newValue, isLoading) {
    //wait until there is a valid record in the field
    if (!newValue)
    return;
```

```
    g_form.getReference('caller_id', vipCallerCallback);
}

function vipCallerCallback(caller) {
     //get the caller object so we can access fields
     var callerLabel = gel('label.incident.caller_id');
     var callerField = gel('sys_display.incident.caller_id');

    //check for VIP status
    if (caller.vip=='true') {

        //change the caller label to red background

   //style object is CSSStyleDeclaration, style names are not standard css names
   if (callerLabel) {
   callerLabel.style.backgroundImage =    'url(images/icons/vip.gif)';
    callerLabel.style.backgroundRepeat            =    'no-repeat';
    callerLabel.style.backgroundPosition = '95% 55%';
    //change the caller's name field to red text
    if (callerField)
            callerField.style.color='red';

   }
   }
   else {
 //not a VIP, remove temporary styles
if (callerLabel)
        callerLabel.style.backgroundImage = '';

     if (callerField)
        callerField.style.color='';

   }
}
```

It's still mostly readable, but let's see if we can make it clearer.

There are three main pieces to the script, and only 1 function:

1. Get the Caller's reference from the database (make a server call)
2. Once the server responds, check the VIP flag
3. If the person is a VIP, flag them
4. Otherwise, remove the flag, if any

Above, the **vipCallerCallback** function is where steps 2, 3, and 4 are taking place (28 lines of code). I don't really need to see the code for steps 3 and 4 unless something isn't working. I really just care about the logic, and what the code *should* be doing.

Once again, with modularization (I've highlighted the function calls for easier tracing):

```
function onChange(control, oldValue, newValue, isLoading) {
    //wait until there is a valid record in the field
    if (!newValue) {
        return;
    }
   g_form.getReference('caller_id', vipCallerCallback);
}

function vipCallerCallback(caller) {
    //get the caller field object so we can access fields
    var callerLabel = gel('label.incident.caller_id');
    var callerField = gel('sys_display.incident.caller_id');

    //check for VIP status
    if (caller.vip=='true') {
        setVIPStyles(callerLabel, callerField);
    }
    else {
        removeVIPStyles(callerLabel, callerField);
    }
}

function setVIPStyles(label, field){
     if (label) {
        label.style.backgroundImage    = 'url(images/icons/vip.gif)';
        label.style.backgroundRepeat   = 'no-repeat';
        label.style.backgroundPosition = '95% 55%';
     }
     //change the caller's name field to red text
     if (field) {
        field.style.color='red';
     }
}

function removeVIPStyles(label, field){
     if (label) {
        label.style.backgroundImage = '';
     }
     if (field) {
        label.style.color='';
     }
}
```

Now the **vipCallerCallback** function is only 10 lines of code. The general coding approach has been left unchanged, but the amount of code required to be read in order to understand the logic has been reduced. You can now reuse the setStyles function repeatedly.

If/Else if/Else

The if/then switch is the primary building block of programming, and most of your time will probably be spent trying to boil down your objective into a small set of absolute rules.

Once you have these rules, you may still need to diagram the conditions, flowchart style, so you'll have a visual aid. But for now, the basics.

Comparison Operators

The key to asking a question (if…) is comparing two things. You will need to study and remember these operators as they will be used over and over again in your scripting:

==	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
===	Equal to, and the same data type (Identical)
!==	Not identical

Once you are comfortable with using the operators above, you can start comparing values with reckless abandon. Let's try it!

```
if ("a" == "A"){
   alert('The As have it!'); // won't alert
}
if (0 <= "1"){
   alert('Naught it is!'); // will alert
}
```

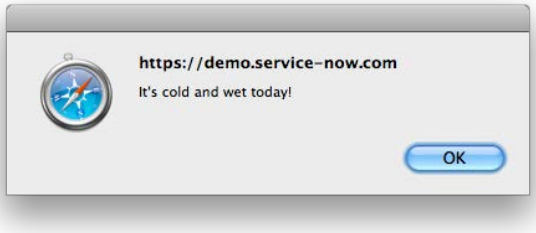

```
if (0 === "0"){
   alert('these are not identical!'); // won't alert
}
```

Compound Comparisons

The above is all well and good, but what happens if you want to compare multiple items at one time?

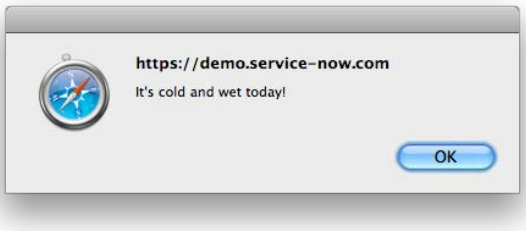
Well, we could do the following, and nest if statements:

```
var precipitation = true;
var temperature = 55;

if (precipitation == true){
   if (temperature <= 60){
      alert("It's cold and wet today!");
   }
}
```



This looks a little messy, and if you have a lot of conditions, it will quickly get hard to read. What we can do is apply Boolean logic, use AND ("&&") and OR ("||") operators to combine conditions to this result:

```
var precipitation = true;
var temperature = 55;

if (precipitation == true && temperature <= 60){
   alert("It's cold and wet today!");
}
```



If/Else if/Else

Now that we understand comparisons, we can make them much more useful by adding an otherwise clause – the "else" clause.

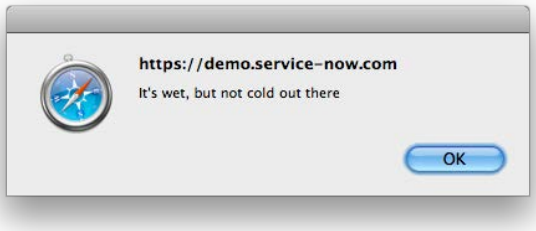
Above, we are checking to see if the weather is cold and wet, but we're not doing anything if the weather is NOT cold and wet.

In our weather info, we have essentially 4 potential results:

1. Both cold and wet
2. Just cold
3. Just wet
4. Neither

By adding an "else" clause, you can continue to check conditions until you find the right one without having to check ALL the conditions, every time.

```
var precipitation = true;
var temperature = 65;

if (precipitation == true && temperature <= 60){
    alert("It's cold and wet today!");
}
else{
    if(precipitation == false && temperature <= 60){
        alert("It's cold but dry today");
    }
    else{
        if(precipitation == true && temperature > 60){
            alert("It's wet, but not cold out there");
        }
        else{
            alert("We've looked at the conditions, and it's neither cold nor wet!");
        }
    }
}
```



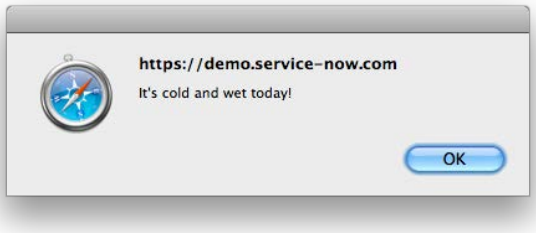
It's a little better but hard to follow – can we simplify it even further?

Introducing: the "else if" statement.

With the else if statement, we can develop a multi-position switch of sorts, only checking the conditions we need:

```
var precipitation = true;
var temperature = 55;

if (temperature <= 60 && precipitation == true){
    alert("It's cold and wet today!");
}
else if (temperature <= 60 && precipitation == false){
    alert("It's cold but dry today");
}
else if(precipitation == true){
    alert("It's wet, but not cold out there");
}
else{
    alert("We've looked at the conditions, and it's neither cold nor wet!");
```

```
}
```



Since our conditions are pretty simple and we only have two on/off switches, we can logically omit the temperature check from the third and fourth conditionals.

Review

Syntax

- Everything is case-sensitive (e.g. Javascript is not the same as JavaScript)
- As long as the starting matches the ending, you can use "double quotes" or 'single quotes'
- Parentheses are used for checking conditions, and sending or receiving a value to/from a function
- Put reminders to a future you by using comments. Use "//" for a single line, or "/*" and "*/" to comment out large blocks of code

Variables

- A variable lets you set it up once to hold a value and then allows you to manipulate its value all throughout the script
- Variables can have different data types – the common ones are:
  1. String (string of characters)
  2. Number
  3. A true or false value
  4. Array (a single object that holds a number of values)
  5. Object (like an array, but with named elements instead of numbered)

Functions

- Let you make pieces of code reusable
- Make your code more legible
- Need a name, arguments (variables to act upon), and a block of code that act upon the inputs
- Use the "return" statement to send the result back to the place that initially called the code

If/Else if/Else

- Compares values, and evaluates them to just "true" or "false" for the sake of running (or not running) a block of code
- Can check many values at a time, using && (AND) and || (OR) operators
- Must have an "if" statement, and can have zero or more "else if" statements, and may have one else clause

Exercises

Exercise 1 – Variables

- Create 3 variables of different object types

Exercise 2 – Functions

- Create a function (name it what you like) that takes 2 numbers, adds them together, and multiplies that by forty-two, and returns the result

Exercise 3 – If/Else if/Else

- Using the "weather" example given, write a simple if/else if/else check to determine if a coffee cup is larger than 12 fl. oz. and is currently hot or piping hot

For further challenges and learning, check out the following resources below. W3Schools is especially powerful as it allows you to write and test code right in the browser so that you can see how changes to the code affect the result!

Resources

JavaScript Books:

- "Simply JavaScript" by Kevin Yank & Cameron Adams, SitePoint
- "JavaScript Pocket Reference" by David Flanagan, O'Reilly
- "Head First JavaScript" by Michael Morrison, O'Reilly
- "JavaScript: Definitive Guide" by David Flanagan, O'Reilly

New to JavaScript:

- For the Non-Programmer: http://webteacher.com/javascript
- JavaScript Primers: http://www.htmlgoodies.com/primers/jsp/
- W3Schools JavaScript Tutorial: http://www.w3schools.com/js/default.asp
- Mozilla Developer Network Doc Center: A re-introduction to JavaScript: https://developer.mozilla.org/en/JavaScript/A_re-introduction_to_JavaScript

Online Resources:

- Online JavaScript Reference: https://developer.mozilla.org/en/JavaScript/Guide
- About JavaScript: https://developer.mozilla.org/en/About_JavaScript
- Prototype Object Model: https://developer.mozilla.org/en/JavaScript/Guide/Details_of_the_Object_Model#Class-_Based_vs._Prototype-Based_Languages
- Google Code University: HTML, CSS and JavaScript from the Ground Up: http://code.google.com/edu/submissions/html-css-javascript/
- About.com JavaScript Tutorials: http://javascript.about.com/od/hintsandtips/Javascript_Tutorials.htm
- Webucator JavaScript Tutorial: http://www.learn-javascript-tutorial.com/