Learn JavaScript on the Now Platform

Contents

Background Information & Resources

JavaScript on the Now Platform enables you to create more complex logic and user experiences. This video series introduces you to the JavaScript language specifically tailored to its use in the Now Platform. Lab exercises in this series give you experience from basic syntax to flow control to complex objects. You'll also Learn how to debug and common best practices all within the Now Platform.

Full playlist of all lesson videos

Overview/introductory video

Repository of all scripts used in the series

Overview post on the ServiceNow Community about the series

Lesson 1: Getting Started

An introduction to what this video series is about (and not about), who it is for, and how to get started. This video also includes examples that you can download and test for yourself to get started writing JavaScript on the Now Platform.

Link to lesson 1 video

Link to lesson 1 code repo

Get a PDI from https://developer.servicenow.com

To run scripts in ServiceNow, go to: **System Definition > Scripts - Background**

1.1 First script

```
// Anything following '//' is a comment

gs.info ('Hello, world!');          // gs.info() = informational output
```

1.2 Example of server-side script

This code does a database query for all the active task records, then prints the task number for each.

```
var gr = new GlideRecord('task');
gr.addActiveQuery();
gr.query();

while (gr.next()) {
    gs.info(gr.getValue('number'));
}
```

1.3 Example of a client-side script

To create a new client script that displays an alert whenever an incident form is loaded:

- Access client scripts at: **System Definition > Client Scripts**
- Click ‘New’ to create a new script
- Give it a name (e.g. Display current state)
- Assign a table (in this case, Incident)
- Type is: onload (because that’s what the script template specifies)
- Add code:

```
function onLoad() {
  alert('Current state value is: ' + g_form.getValue('state'));
}
```

- Click ‘Update’
- The alert will now display the current state whenever an incident form is loaded (e.g., via Incident > All)

Lesson 2: Statements and syntax

This lesson covers the basics of JavaScript syntax and when semicolons should/shouldn't be used.

Link to lesson 2 video

Link to lesson 2 code repo

2.1 When semicolons are required

```
var i = 0; i++              // <-- semicolon obligatory
                            // (but optional before newline)
var i = 0                   // <-- semicolon optional
    i++                     // <-- semicolon optional
```

2.2 When semicolons are optional

```
var i;                              // variable declaration
i = 5;                              // value assignment
i = i + 1;                          // value assignment
i++;                                // same as above
var x = 9;                          // declaration & assignment
var fun = function() {...};         // var declaration, assignment, and func. definition
alert("hi");                        // function call
```

2.3 When to avoid semicolons

```
NO semicolons after '}':

if (...) {...} else {...}
for (...) {...}
while (...) {...}

BUT:

do {...} while (...);
```

Function statements:

```
function (arg) { /*do this*/ }                  // NO semicolon after ‘}’

Exception:
for (var i=0; i < 10; i++) {/*actions*/}        // Correct
for (var i=0; i < 10; i++;) {/*actions*/}       // This will cause a syntax error
```

Lesson 3: Variables

This lesson demonstrates some simple scalar (or primitive) variable types, how to declare and set them and best practices for naming your variables.

Link to lesson 3 video

Link to lesson 3 code repo

3.1 Simple variables & good/bad variable names

```
var name = 'Chuck';                 // Simple string variable
var i = 0;                          // Simple integer variable
var answer = true;                  // Simple Boolean variable
```

3.2 Naming examples

Examples of what not to do:

```
var c = "http://www.amazon.com";                  // ‘c’ is not descriptive enough
var case = 'CASE0010001';                         // ‘case’ is a reserved word
var lastEntryInTheListWithRelatedRecords = true;  // Too long
```

Rather than variable names such as ‘person’, it’s best to put the variable type on the end of the var name, e.g.:

```
personCount                 // Indicates a counter/integer
personList                  // Must be a list
personObj                   // Object
personGr                    // GlideRecord
```

Lesson 4: Simple Arithmetic Operators

This lesson covers some simple arithmetic operators to help you understand how to add, subtract, multiply, divide, and modulo (the remainder of division). It also includes some time saving shortcuts.

Link to lesson 4 video

Link to lesson 4 code repo

```
// Assignment
var a = 12;
var b = 3;

// Addition
gs.info(2 + 2);
gs.info(a + 2);
b = b + 2;
// b += 2;     // shorthand for the line above
gs.info(a + b);

// Increment by 1
a++;  /* or '++a;' is the same thing */  gs.info(a);

// Decrement by 1
b--; gs.info(b);

// Multiply
gs.info(a * b);

// Division
gs.info(a / b);

// Modulo - get the remainder of a division
gs.info('');
gs.info(a);
gs.info(b);
gs.info(a % b);

var c = (5 + 4) * 2;
gs.info(c);               //  output is 18
```

Lesson 5: Common Error Messages

Lesson 5 covers some common error messages you may encounter while writing JavaScript and how to overcome them.

Link to lesson 5 video

Link to lesson 5 code repo

Script containing errors:

```
gs.info(myUnknownVariable);
ga.info('Hello, world!);
```

Lab 2

This lab exercise challenges you to test your arithmetic skills to get the right answer.

Instructions:

What is 3 + 2 * 5?

Link to answer video

Solution:

```
gs.info(3 + 2 * 5);      // Returns: 13
```

Lesson 6: Strings

Introducing strings, the length property, and concatenating strings together.

Link to lesson 6 video

Link to lesson 6 code repo

```
var firstName = "Chuck";
var lastName  = 'Tomasi';  // Either is ok
var myCar     = "Chuck's Car";  // Mix them to avoid issues

// Note the use of + when strings are involved

var name = firstName + ' ' + lastName;
gs.info(name);
gs.info('length of name=' + name.length);
```

Lesson 7: Special Characters

How to deal with special characters like tabs, newlines, quotes, and backslashes that can make your strings tricky.

Link to lesson 7 video

Link to lesson 7 code repo

```
// \n = new line
// \t = tab
// \\ = backslash
// \' = single quote
// \" = double quote

gs.info('Single string\nTwo lines');
gs.info('Chuck\'s simple script');
gs.info('Don\'t confuse a forward slash (/) with a backslash(\\)');
```

Lesson 8: Data Type Conversions

Lesson 8 shows you how to convert an integer to a string, a string to an integer, and how to determine what type of variable you have if you're not sure.

Link to lesson 8 video

Link to lesson 8 code repo

```
var i = 5;
var iStr = i.toString();            // convert an integer to a string

gs.info('type of I = ' + typeof i);
gs.info('type of iString = ' + typeof iStr);       // variable type

var n = parseInt(iStr);             // convert a string into a number
gs.info(typeof n + ' n=' + n);

var n = parseFloat(iStr);           // convert a string into a floating point number
```

Lab 3: Instructions

Create three string variables. Print the length of each string. concatenate them together with a new line character and save to a new variable. Print the new variable and length of the new variable.

Lab 3: Solution

This video demonstrates the answer to the lab assignment given in lesson 8.

Link to Lab 3 solution video

Instructions:

Create three string variables. Print the length of each string. concatenate them together with a new line character and save to a new variable. Print the new variable and length of the new variable.

Solution:

```
var string1 = 'Chuck';

var string2 = 'Tomasi';

var string3 = 'JavaScript';

gs.info('length1 = ' + string1.length);

gs.info('length2 = ' + string2.length);

gs.info('length3 = ' + string3.length);


var allStrings = string1 + '\n' + string2 + '\n' + string3;

gs.info('allStrings=' + allStrings);

gs.info('length of allStrings=' + allStrings.length);
```

Output:

```
*** Script: length1 = 5

*** Script: length2 = 6

*** Script: length3 = 10

*** Script: allStrings=Chuck

Tomasi

JavaScript

*** Script: length of allStrings=23
```

Lesson 9: Commenting

Lesson 9 takes explains what comments are and why you want to use them and how they can be formatted. It also includes some best practices for commenting.

Link to lesson 9 video

Link to lesson 9 code repo

```
// This is a single line comment

/* This is another way to comment */

/*
 This is a multi-line comment.
 This code is commented out!
 var name = 'Chuck';
 */
```

Lesson 10: Comparisons

Introducing and demonstrating comparisons and logical expressions when comparing to variables.

Link to lesson 10 video

Link to lesson 10 code repo

```
var a = 0;
var b = 1;
gs.info(a < b);       // output: true or false

gs.info(a = b);       // assigns b to a
gs.info(a == b);      // equals, output: true or false
gs.info(a != b);      // does not equal: output: true or false

var n = '3';
var i = 3;
gs.info(n == i);      // Output: true (same value) – because JavaScript is a 'loosely
   typed language'
gs.info(n === i);     // Checks for equality of value AND data type
```

Lesson 11: The If Statement and Boolean Logic

In lesson 11, you learn about the if statement for controlling the flow of your JavaScript. This video also takes a look at basic Boolean logic with AND, OR, and NOT operators to make more complex logical comparisons.

Link to lesson 11 video

Link to lesson 11 code repo

11.1 If statements

```
var a = 1; var b = 3; var c = 5;
if (a < b)
gs.info('a is less than b');
var bool = a < b;
if (bool) {
gs.info ('a is less than b');
} else { … }

//  else
if (a < b)
  gs.info('a is less than b');
else
  gs.info('a is greater than or equal to b');

// Else if and else
if (a < b)
  gs.info('a is less than b');
else if (a > b)
  gs.info('a is greater than b');
else if (a == b)
  gs.info('a equals b');
else
  gs.info('Uh-oh');

if (a < b)
  if (b < c)
      gs.info('a b c are in order');
```

11.2 Boolean logic tables

```
// AND (&&) - both must be true

//      +-------------+-------------+-------------+

//      |   AND       |  true       |  false      |

//      +-------------+-------------+-------------+

//      |  true       |  true       |  false      |

//      +-------------+-------------+-------------+

//      |  false      |  false      |  false      |

//      +-------------+-------------+-------------+


// OR (||) - Either must be true

//      +-------------+-------------+-------------+

//      |   OR        |  true       |  false      |

//      +-------------+-------------+-------------+

//      |  true       |  true       |  true       |

//      +-------------+-------------+-------------+

//      |  false      |  true       |  false      |

//      +-------------+-------------+-------------+


// NOT (!) - reverse the logic
//      +-------------+-------------+-------------+
//      |   NOT       |  true       |  false      |
//      +-------------+-------------+-------------+
//      |             |  false      |  true       |
//      +-------------+-------------+-------------+
```

11.3 Boolean logic code examples

```
var a = 1;
var b = 3;
var c = 5;

if (a < b && b < c)
  gs.info ('a b c are in order');

if (b > a || b > c)
  gs.info ('b is greater than one of them.');

// Note, indentation can be deceptive!!!
if (a < b)
  if (b < c) {
    gs.info('a b c are in order');
    gs.info(' that means a is less than c');
  }
else
  gs.info('a is greater than or equal to b');

var valveOpen = true;
if (valveOpen == true)
  gs.info('Valve is currently open');

if (bool)
  gs.info('Valve is currently open');

var valveOpen = false;
if (!valveOpen)
  gs.info('Valve is currently closed');
```

Lesson 12: The Ternary Operator

How to create a short method of the if/else statement.

Link to lesson 12 video

Link to lesson 12 code repo

Could have written:

```
var valveOpen = true;
var openStatusString;
if (valveOpen)
  openStatusString = 'open';
else
  openStatusString = 'closed';
  gs.info('1: Valve is currently ' + openStatusString);
```

Introducing a shortcut way the ternary operator):

```
var openStatusString = (valveOpen) ? 'open' : 'closed';
gs.info('2: Valve is currently ' + openStatusString);
```

Lesson 13: The Switch Statement

The switch statement is a cleaner way to do if/else if/else if/else on the same variable.

Link to lesson 13 video

Link to lesson 13 code repo

Using if/else if/else if:

```
var level = 5;
var message = '';
if (level == 0)
  message = 'Empty';
else if (level == 1 || level == 2)
  message = 'Low';
else if (level == 3)
  message = 'Medium';
else if (level == 4)
 message = 'High';
else if (level == 5)
 message = 'Full';
else
  message = 'Uh-oh';
gs.info('Level=' + level + ' status=' + message);
```

Using the switch statement:

```
var level = 5;
var message = '';
switch (level) {             //can only be an integer or a string
  case 0:
    message = 'Empty';
    break;

  case 1:

  case 2:
    message = 'Low';
    break;

  case 3:
    message = 'Medium';
    break;

  case 4:
    message = 'High';
    break;

  case 5:
    message = 'Full';
    break;

  default:
    message = 'Uh-oh!';
}

gs.info('Level=' + level + ' status=' + message);
```

Lesson 14: Truthy/Falsy

Lesson 14 takes a look at what makes Boolean, integer, and even string variables true or false in an condition.

Link to lesson 14 video

Link to lesson 14 code repo

Simple case of true and false:

```
var boolTrue = true;
var boolFalse = false;
gs.info ('boolTrue=' + boolTrue + ' boolFalse=' + boolFalse);
```

What about numbers?

```
var num = 0;          // <== try with different numbers

gs.info (num + ' is ' + ((num) ? 'true' : 'false'));

                      // Only returns false for 0, all other results are true
```

Take a look at strings:

```
var string1;                        // undefined, so gs.info would produce an error
var string2 = null;                 // an empty string, but not undefined
var string3 = 'Hello, world!';

gs.info ('string1=' + ((string1) ? 'true' : 'false'));           // false
gs.info ('string2=' + ((string2) ? 'true' : 'false'));           // false
gs.info ('string3=' + ((string3) ? 'true' : 'false'));           // true
```

Lab 4: Instructions

Create a script that takes a string variable “Hello world” and translates it to at least 3 different languages based on a 'language' variable.

Make the default language English if a match isn't found.

Lab 4: Solution

Link to Lab 4 solution video

Instructions:

Create a script that takes a string variable “Hello world” and translates it to at least 3 different languages based on a 'language' variable.

Make the default language English if a match isn't found.

Solution:

```
var language   = 'German';
var fromString = 'Hello world';
var toString   = '';

switch (language) {
    case 'German':
        toString = 'Hallo Welt';
        break;
    case 'French':
        toString = 'Bonjour le monde';
        break;
    case 'Spanish':
        toString = 'Hola Mundo';
        break;
    default:
        toString = 'Hello, world';
}

gs.info(fromString + ' in ' + language + ' ==> ' + toString);
```

Output:

```
Hello world in German ==> Hallo Welt
```

Lesson 15: The While Loop

Link to lesson 15 video

Link to lesson 15 code repo

15.1 Simple while loop

```
var i = 0;
while (i < 5) {
  gs.info(i);
  i++;
}
gs.info('done i=' + i);
```

15.2 Breaking out of a while loop

```
var i = 0;
while (true) {
  if (i == 5)
     break;
  gs.info(i);
  ++i;
}
gs.info('done');
```

15.3 Continue - jumping back to the while condition

```
var i = 0;
var done = false;
while (!done) {
  if (i < 5) {
    ++i;
    gs.info(i + ' done=' + done);
    continue;
  }
  gs.info('I think we are done');
  done = true;
}
gs.info(i);
```

Lesson 16: The For Loop

This lesson explains and demonstrates how to use the for loop to run one or more statements multiple times.

Link to lesson 16 video

Link to lesson 16 code repo

Note: break and continue work here too!

```
for (var i = 0; i < 5; i++) {
  gs.info(i);
}
gs.info('done i=' + i);
```

Lesson 17: The Do-While Loop

Link to lesson 17 video

Link to lesson 17 code repo

```
var i = 0;
gs.info('start');
do {
  gs.info('i=' + i);
  ++i;
} while (i < 5);
gs.info('done i=' + i);
```

Lesson 18: Nested Loops

Explains and demonstrates how to use a loop within a loop. While this example shows two for loops, any loop can be placed inside another. However, there are some considerations to take when using nested loops.

Link to lesson 18 video

Link to lesson 18 code repo

```
for (var i = 0; i < 5; i++) {
  for (var j = 0; j < 3; j++) {
    gs.info('i=' + i + ' j=' + j);
  }
}
gs.info('Done i=' + i + ' j=' + j);
```

Lab 5: Instructions

Create a script that creates 5 teams of 4 people and assigns a unique identifier to each team member.

The script should display the team number, person number, and identifier for that person/team.

Avoid displaying zeros for a better user experience.

Video explaining lab 5 Instructions

Lab 5: Solution

Instructions:

Create a script that creates 5 teams of 4 people and assigns a unique identifier to each team member.

The script should display the team number, person number, and identifier for that person/team.

Avoid displaying zeros for a better user experience.

Link to Lab 5 solution video

Solution:

```
var id = 1;
for (var team = 0; team < 5; team++) {
    for (var person = 0; person < 4; person++) {
        gs.info('id=' + id + ' team=' + team+1 + ' person=' + person+1);
        ++id;
    }
}
```

Lesson 19: Functions

Lesson 19 introduces you to the topic of functions. You will learn when and why you might want to build a function, how to pass information in to a function and how to get information out. It also demonstrates how to call a function from within another function.

Link to lesson 19 video

Link to lesson 19 code repo

19.1 Functions

```
function sayHello() {    // a function without parameters
  gs.info('Hello');
}
sayHello();              // call the function
```

19.2 Function with a parameter

```
function toCelsius(fahrenheit) {

    var c = (5 / 9) * (fahrenheit - 32);

    gs.info(c);
}
toCelsius(32);
toCelsius(100);
```

19.3 Function with a return value

```
function toCelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);     // return – used to get a value out of a function
}
var c = toCelsius(32);
gs.info(c);
c = toCelsius(212);
gs.info(c);
```

19.4 Local variables scope

```
function toCelsius(fahrenheit) {

    // c is only known in the function toCelsius()
    var c = (5 / 9) * (fahrenheit - 32);

    return c;  // c is a local variable within this function
}
gs.info(c);    // Here, c is out of scope – it will return c is undefined
```

19.5 - Global variables and local

```
/ var convertTo = 'F';

function toCelsius(f) {
    var c = (5 / 9) * (f - 32);

    return c;
}

function toFahrenheit(c) {
    var f = c * 9 / 5 + 32;

    return f;
}

function convertTemp(temp) {

  // use the global variable to determine conversion
  if (convertTo == 'C') {
    return toCelsius(temp);
  } else {
    return toFahrenheit(temp);
  }
}
gs.info(convertTemp(100));
```

19.6 Self running function

```
// This code is outside the function
var i = 20;

(function() {
  // Local variable
  i = 10;            // uh-oh, forgot the var!

  gs.info('i=' + i);
}());

i = 3;
gs.info('i=' + i);
```

Lesson 20: Try/Catch/Finally Statements

Lesson 20 shows you how to safeguard your code against rogue errors so it continues to run using the try/catch/finally construct.

Link to lesson 20 video

Link to lesson 20 code repo

20.1 Bad script w/o try/catch

```
/ for (var i = 0; i < 5; i++) {
  gs.info('i=' + i + ' answer=' + answer);
}
gs.info('done');
```

20.2 Trapping that error

```
try {
  for (var i = 0; i < 5; i++) {
    gs.info('i=' + i + ' answer=' + answer);
  }
} catch (e) {
  gs.error ('Uh-oh ' + e.message);   // error output
}                                    // error will also appear in ‘System Logs - All’
gs.info('done');
```

20.3 And finally …

```
try {
  for (var i = 0; i < 5; i++) {
    gs.info('i=' + i + ' answer=' + answer);
  }
} catch (e) {
  gs.error('Uh-oh ' + e.message);
} finally {
  gs.info('done');
}
```

Lesson 21: Simple Database Query

This lesson shows you some basic operations to retrieve records from the database using GlideRecord on ServiceNow. This applies some of the earlier concepts with functionality specific to the Now Platform.

Link to lesson 21 video

Link to lesson 21 code repo

21.1 Get and display numbers on all incidents

```
var incGr = new GlideRecord('incident');
incGr.query();

while (incGr.next()) {                    // remember: (incGr.next()) returns a Boolean
  gs.info(incGr.getValue('number'));
}
```

21.2 Get and display numbers on all incidents v2

```
var incGr = new GlideRecord('incident');
incGr.addQuery('active', true);                  // add a filter
incGr.orderBy('number');                  // add order
incGr.setLimit(5);                        // set a limit
incGr.query();

while (incGr.next()) {
  gs.info(incGr.getValue('number'));
}
```

21.3 What is 'number'?

```
var incGr = new GlideRecord('incident');
incGr.setLimit(1);
incGr.query();

if (incGr.next()) {
  var dotNumber = incGr.number;
  var gvNumber  = incGr.getValue('number');
  gs.info('dotNumber=' + typeof dotNumber + ' gvNumber=' + typeof gvNumber);
}

// Why should I care? I'll show you when we get to arrays...
```

21.4 Getting a single record quickly

Just need one record? Use `.get(SYSID)` or `.get('fieldName', fieldValue)`

```
var incGr = new GlideRecord('incident');

if (incGr.get('965c9e5347c12200e0ef563dbb9a7156')) {

// If you pass only one value, it's assumed to be a sys_id.

  gs.info(incGr.getValue('number'));
}

// Or:

var incGr = new GlideRecord('incident');

if (incGr.get('number', 'INC0000059')) {
  gs.info(incGr.getValue('sys_id'));
}
```

Lesson 22: Introduction to Arrays

Introducing the concept of arrays. Arrays are a great way to keep a list of the same data type (integer, string, etc.) in memory and address them as one unit or individually.

Link to lesson 22 video

Link to lesson 22 code repo

22.1 Making a simple array

```
var list = [];
list[0] = 1;          // The first position is always 0.
list[1] = 3;
list[2] = 5;
gs.info('length of list is: ' + list.length);             // In this case, returns ‘3’
```

Alternatively, make an optional declaration, but this method is not preferred:

```
/ var list = Array();
```

A shorter way:

```
var list = [1, 3, 5];
gs.info('length of list is: ' + list.length);
```

22.2 Loops and arrays

```
var list = [1, 3, 5];
for (var i = 0; i < list.length; i++) {
  gs.info('i=' + i + ' value=' + list[i]);
}
```

A slightly better way:

```
var list = [1, 3, 5];
var len = list.length;
for (var i = 0; i < len; i++) {
  gs.info('i=' + i + ' value=' + list[i]);
}
```

Lesson 23: Array forEach

This lesson demonstrates a unique way to create a loop using an array with the forEach method.

Link to lesson 23 video

Link to lesson 23 code repo

forEach with external function:

```
var list = [1, 3, 5];
list.forEach(myFunction);
function myFunction(item, i) {
  gs.info('myFunction item=' + item + ‘ i=’ + i);
}
```

Embedded forEach:

```
var list = [1, 3, 5];
list.forEach(function (item) {
  gs.info('embedded function item=' + item);
});
```

Also returning the entire array:

```
var list = ['apple', 'banana', 'orange'];
list.forEach(function (item, index, arr) {
  gs.info('embedded function item=' + item + ' index=' + index + ' arr=' + arr);
});
```

Lesson 24: Common Array Methods

Lesson 24 demonstrates some of the common things you can do with arrays to manage the elements they contain.

Link to lesson 24 video

Link to lesson 24 code repo

1. Common array methods/functions

```
var list = ['Chuck', 'Kreg', 'Stacey'];
gs.info('list=' + list);
```

or:

```
gs.info('list=' + list.toString());         //converts to string
```

2. join(string)

```
var list = ['Chuck', 'Kreg', 'Stacey'];
gs.info('join: list=' + list.join(', '));
```

Or:

```
gs.info('join: list=' + list.join(\n));
// to insert a new line between each item in the list
```

push(value1, value2, ..., valueX)

```
list.push('Dave');
list.push('Andrew');
gs.info('push: list=' + list.join(', '));
```

pop()

The last thing pushed onto the stack is the first thing popped off.

```
var list = ['Chuck', 'Kreg', 'Stacey'];
gs.info('join: list=' + list.join(', '));
list.push('Dave');
list.push('Andrew');
gs.info('push: list=' + list.join(', '));
list.pop();
gs.info('pop: list=' + list.join(', '));
```

Output:

```
*** Script: join: list=Chuck, Kreg, Stacey
*** Script: push: list=Chuck, Kreg, Stacey, Dave, Andrew
*** Script: pop: list=Chuck, Kreg, Stacey, Dave
```

3. shift()

```
var list = ['Chuck', 'Kreg', 'Stacey'];
gs.info('Before shift(), list[0]=' + list[0]);
list.shift();
gs.info('shift: list=' + list.join(', '));
gs.info('After shift(), list[0]=' + list[0]);
```

Output:

```
*** Script: Before shift(), list[0]=Chuck
*** Script: shift: list=Kreg, Stacey
*** Script: After shift(), list[0]=Kreg
```

4. unshift

Puts things on the front of the array.

```
var list = ['Chuck', 'Kreg', 'Stacey'];
var newLength = list.unshift('Jason', 'Andrew');
gs.info('new length=' + newLength + ' unshift() list=' + list.join(', '));
```

Output:

```
*** Script: new length=5 unshift() list=Jason, Andrew, Chuck, Kreg, Stacey
```

5. splice

To add/remove elements from somewhere in the middle of the array.

Notation:

```
splice (starting-position, remove-n-elements, add-value1, add-value2, ..., add-valueX)
```

```
var names = ["Eric", "Donna", "Melanie", "Jessie"];
gs.info(names.join(', '));
names.splice(2, 0, "Cary", "Henri"); // Start at posn. 2, remove none, add 2
gs.info(names.join(', '));
```

Output:

```
*** Script: Eric, Donna, Melanie, Jessie
*** Script: Eric, Donna, Cary, Henri, Melanie, Jessie
```

6. slice

Extract part of an array into another array.

Notation:

```
slice(start, end)
```

```
var names = ["Eric", "Donna", "Melanie", "Jessie", "Howard", "Tomasz"];
gs.info(names.join(', '));
var subNames = names.slice(1, 3);    // Get names at positions 1 and 2
// (i.e. 'between' the start of 1 & the start of 3)
gs.info(subNames.join(', '));
```

Output:

```
*** Script: Eric, Donna, Melanie, Jessie, Howard, Tomasz
*** Script: Donna, Melanie
```

7. Reverse the elements of an array

```
var names = ["Eric", "Donna", "Melanie", "Jessie", "Howard", "Tomasz"];
names.reverse();
gs.info(names.join(', '));
```

Output:

```
*** Script: Tomasz, Howard, Jessie, Melanie, Donna, Eric
```

8. Getting the value

```
var list = [];
var countReturned = 0;
var fName = ‘sys_id’;
var incGr = new GlideRecord('incident');
incGr.addQuery('priority', '1');
incGr.query();

while (incGr.next()) {
    list.push(incGr.getValue(fName));
++ countReturned;
}

gs.info('list=\n' + list.join('\n'));
gs.info(countReturned);
```

Output:

```
*** Script: list=
000d695897311110f130f5b3f153afa9
0034d4f035ad7410f877125fed0fc03b
0039936a977e5110f130f5b3f153af19
Etc. …
55738783688d6010f877d65b9fbe09b6
*** Script: 3244
```

Lesson 25: ServiceNow ArrayUtil

Lesson 25 builds on lesson 24 with some additional array utilities provided by ServiceNow.

Link to lesson 25 video

Link to lesson 25 code repo

ArrayUtil is part of a ServiceNow Script Include. To review the script, etc. go to: **System Definitions > Script Includes, search term: '=ArrayUtil'**

```
var au = new ArrayUtil();
var names = [
  "Eric",
  "Donna",
  "Melanie",
  "Jessie",
  "Howard",
  "Eric",                                  // Note 'Eric' is listed twice
  "Jessie",
  "Tomasz"
];
var newNames = au.unique(names);
            // One of the capabilities of ArrayUtil is to check for uniqueness.
gs.info(newNames.join(', '));              // Good for de-duping arrays
```

Output:

```
*** Script: Tomasz, Howard, Jessie, Melanie, Donna, Eric
```

Lab 6a: Instructions

Create a script to accept a table name and return a list records display values.

Hint: use GlideRecord.getDisplayValue()

Lab 6a: Solution

Instructions:

Create a script to accept a table name and return a list records display values.

Hint: use GlideRecord.getDisplayValue()

Link to Lab 6a solution video

Solution:

```
// @param tableName - name of table to query
// @return array - list of record display values

function listRecords(tableName) {

    var answer = [];
    var recGr = new GlideRecord(tableName);
    recGr.query();

    while (recGr.next()) {
        answer.push(recGr.getDisplayValue());
    }

    return answer;
}

gs.info(listRecords('incident').join('\n'));
```

Lab 6b: Instructions

Update your previous script to accept a limit parameter.

Hint: use GlideRecord.setLimit()

Lab 6b: Solution

Instructions:

Update your previous script to accept a limit parameter.

Hint: use GlideRecord.setLimit()

Link to answer video

Solution:

```
// @param tableName - name of table to query
// @param limit - integer > 0
// @return array - list of record display values

function listRecords(tableName, limit) {

    var answer = [];
    var recGr = new GlideRecord(tableName);
    if (limit && limit > 0) {
        recGr.setLimit(limit);
    }
    recGr.query();

    while (recGr.next()) {
        answer.push(recGr.getDisplayValue());
    }

    return answer;
}

gs.info(listRecords('incident', 10).join('\n'));
```

Lesson 26: Introduction to Objects

Introducing the concept of JavaScript objects: what they are, and how to create and manage them.

Link to lesson 26 video

Link to lesson 26 code repo

26.1 Simple Objects

```
var box = {};              // Alternative syntax: var box = new Object();
box.height   = 20;
box.width    = 10;
box.length   = 10;
box.material = "cardboard";
box.open     = true;
gs.info(box.material);
```

26.2 Bracket notation

An alternative syntax for defining object properties:

```
var vehicle = {};
vehicle['year'] = 2018;
vehicle['make'] = "Toyota";
vehicle['model'] = "Sienna";
gs.info(vehicle['year'] + ' ' + vehicle['make'] + ' ' + vehicle['model']);
```

26.3 Shortcut: JSON format

Initializing and setting name/value pairs at the same time: using JSON (JavaScript Object Notation) format declaration:

```
var vehicle = {
  "year" : 2018,        // It's best to use double quotes, although single quotes may work
  "make" : "Toyota",
  "model" : "Sienna"
};
gs.info(vehicle['year'] + ' ' + vehicle['make'] + ' ' + vehicle['model']);
```

Lesson 27: Checking if an Object has a Property

How to check if an object has a given property and why this would be useful to know.

Link to lesson 27 video

Link to lesson 27 code repo

'hasOwnProperty' method

```
var vehicle = {
  "year" : 2018,
  "make"  : "Toyota",
  "model" : "Sienna"
};
gs.info(vehicle.hasOwnProperty("year"));    // <== true
gs.info(vehicle.hasOwnProperty("price"));   // <== false
```

Lesson 28: Finding All Properties in an Object

How to find all properties within an object using a slightly different version of a for loop than you may have seen before.

Link to lesson 28 video

Link to lesson 28 code repo

Get object keys:

```
var vehicle = {
  "year" : 2018,
  "make"  : "Toyota",
  "model" : "Sienna"
};
for (var key in vehicle) {
  gs.info('key=' + key + ' value=' + vehicle[key]);
}
```

Lesson 29: Arrays of Objects

This lesson combines the lessons on arrays AND objects to create arrays OF objects and why they are so useful.

Link to lesson 29 video

Link to lesson 29 code repo

```
var bookList = [

// This is an array containing 3 objects; each object has 2 properties

  {
    "title" : "Harry Potter and the Chamber of Secrets",
    "author" : "J.K. Rowling"
  },
  {
    "title" : "Moby Dick",
    "author" : "Herman Melville"
  },
  {
    "title" : "A Tale of Two Cities",
    "author" : "Charles Dickens"
  }
];
var len = bookList.length;                          // Tells us: how many books are in the library.
gs.info('Last author=' + bookList[len - 1].author);       // 'len - 1' because len is 3 but array                  // numbering start at 0 and stops at 2
for (var i = 0; i < len; i++) {
  var book = bookList[i];
  gs.info(i + ' - Title: ' + book.title + ' - Author: ' + book.author);
}
```

Lesson 30: JSON Stringify and Parse

Lesson 30 introduces JSON.stringify() and JSON.parse() to translate objects to strings and back again. Watch the video to find out why this is such a powerful JavaScript feature.

Link to lesson 30 video

Link to lesson 30 code repo

30.1 Stringify and Parse

```
var bookList = [
  {
    "title" : "Harry Potter and the Chamber of Secrets",
    "author" : "J.K. Rowling"
  },
  {
    "title" : "Moby Dick",
    "author" : "Herman Melville"
  },
  {
    "title" : "A Tale of Two Cities",
    "author" : "Charles Dickens"
  }
];
gs.info(bookList);
```

Output:

```
/// *** Script: [object Object],[object Object],[object Object]
// That's not very helpful
```

```
var bookListStr = JSON.stringify(bookList);
gs.info(bookListStr);
```

Output:

```
// *** Script: [{"title":"Harry Potter and the Chamber of Secrets","author":"J.K.
Rowling"},{"title":"Moby Dick","author":"Herman Melville"},{"title":"A Tale of Two
Cities","author":"Charles Dickens"}]
// I can read it - sort of.
```

```
var bookListStrFormat = JSON.stringify(bookList, null, 4);
gs.info(bookListStrFormat);
```

Output:

```
//* *** Script: [
    {
        "title": "Harry Potter and the Chamber of Secrets",
        "author": "J.K. Rowling"
    },
    {
        "title": "Moby Dick",
        "author": "Herman Melville"
    },
    {
        "title": "A Tale of Two Cities",
        "author": "Charles Dickens"
    }
] */
// Ah - that's better!
```

30.2 Parse example

```
var libraryStr = '[{"title":"Harry Potter and the Chamber of Secrets","author":"J.K.
Rowling"},{"title":"Moby Dick","author":"Herman Melville"},{"title":"A Tale of Two
Cities","author":"Charles Dickens"}]';
// Enclose string in single quotes, not double quotes


gs.info('length=' + libraryStr.length);
var libraryObj = JSON.parse(libraryStr);
gs.info('length=' + libraryObj.length);
gs.info(JSON.stringify(libraryObj, null, 4));   // 4 spaces of indentation, see link below
```

[See MDN Web Docs reference to JSON.stringify()]

Output:

```
*** Script: length=186
*** Script: length=3
*** Script: [
    {
        "title": "Harry Potter and the Chamber of Secrets",
        "author": "J.K. Rowling"
    },
    {
        "title": "Moby Dick",
        "author": "Herman Melville"
    },
    {
        "title": "A Tale of Two Cities",
        "author": "Charles Dickens"
    }
]
```

Lab 7: Instructions

Update your previous lab to return the sys_id and display value of the records you found using an array of objects. Return an array of objects.

Lab 7: Solution

Instructions:

Update your previous lab to return the sys_id and display value of the records you found using an array of objects. Return an array of objects.

Link to Lab 7 solution video

Solution:

```
// @param tableName - name of table to query
// @param limit - integer > 0
// @return array of objects
//    {
//      "display_value" : <display value>,
//      "sys_id" : <sys_id>
//    }
//
function listRecords(tableName, limit) {

    var answer = [];
    var recGr = new GlideRecord(tableName);
    if (limit && limit > 0) {
        recGr.setLimit(limit);
    }
    recGr.query();

    while (recGr.next()) {
        var obj = {};
        obj.display_value = recGr.getDisplayValue();
        obj.sys_id = recGr.getUniqueValue();
        answer.push(obj);
    }

    return answer;
}

var list = listRecords('incident', 10);
gs.info(JSON.stringify(list, null, 4));
```

Lesson 31: More String Methods

A look at more string methods to help manage this fundamental data type in JavaScript.

Link to lesson 31 video

Link to lesson 31 code repo

31.1 Find the position of a character or substring

```
var subject = 'Warning: Email is not working';
var pos = subject.indexOf('Email');
gs.info(pos);
// returns -1 if no match, 0 if match is at start of the string
```

31.2 Use the position of a character/substring as a condition

```
var short_description = 'System is displaying error message';
if (short_description.indexOf('error') >= 0) {
  gs.info("Error message found");
} else
  gs.info("Error message not found");
}
```

31.3 Get a substring

```
var str = 'Approved: RITM0010001 - Laptop renewal';
var pos = str.indexOf('RITM');
var ritmNumber = str.substring(pos, pos + 11);
gs.info(ritmNumber);
```

31.4 Note: case matters with strings

```
var firstName = 'Chuck';
var loginName = 'chuck';
if (loginName == firstName) {
  gs.info('names match');
} else {
  gs.info('names do not match');
}
```

31.5 Using toUpper or toLower for better matching

```
var firstName = 'Chuck';
var loginName = 'chuck';
gs.info('firstName=' + firstName.toUpperCase() + ' loginName=' + loginName.toUpperCase());
if (loginName.toUpperCase() == firstName.toUpperCase()) {
  gs.info('names match');
} else {
  gs.info('names do not match');
}
```

Lesson 32: Recursion

Recursion is the concept of a function calling itself. While this may seem like a crazy idea, it is a very powerful tool when dealing with hierarchical data like business services. However, there are some things about recursion you need to pay attention to, or you could find yourself in an infinite loop.

Link to lesson 32 video

Link to lesson 32 code repo

An example of where recursion is used is anywhere there is a parent-child relationship with records (e.g. tasks, CIs, etc.) You can recurse "up" the CI tree to find the parent service, or recurse "down" to start with a service and ensure you find all the child/related CIs.

It's a similar concept to the way Service Watch works for business service discovery.

The following function calculates the factorial of x (often written as 'x!') e.g. 5! = 5 x 4 x 3 x 2 x 1 = 120

```
function factorial(x) {

  // TERMINATION
  if (x < 0)
    return;

  // BASE
  if (x === 0)        // === not only compares the value, but also the type.
                      // For example: "1" == 1 is true, whereas "1" === 1 is false
                      // because a string and an integer are different types.
    return 1;

  // RECURSION
  return x * factorial(x - 1);

}

gs.info(factorial(3));              //  Returns 6
```

Lesson 33: Classes

How to improve testing and reduce maintenance by creating object templates known as classes. Go beyond properties and add functions to your objects!

Link to lesson 33 video

Link to lesson 33 code repo

33.1 Classes, Objects, and Prototypes

```
var person = Class.create();
person.prototype = {
    initialize: function() {       // This is the standard formatting for a script include
      this.firstName = '';
      this.lastName  = '';
    },
    setFirstName : function(str) {
      this.firstName = str;
    },

    setLastName : function(str) {
      this.lastName = str;
    },

    getDisplayName : function() {
      return this.firstName + ' ' + this.lastName;
    },

    type: person
};

var me = new person();
me.setFirstName('Chuck');
me.setLastName('Tomasi')
gs.info('me=' + me.firstName + ' ' + me.lastName); // Not advised
var name = me.getDisplayName();
gs.info(name);
```

33.2 Initialize values

```
var person = Class.create();
person.prototype = {
    initialize: function(str1, str2) {
      this.firstName = str1;
      this.lastName  = str2;
    },

    setFirstName : function(str) {
      this.firstName = str;
    },

    setLastName : function(str) {
      this.lastName = str;
    },

    getDisplayName : function() {
      return this.firstName + ' ' + this.lastName;
    },

    type: 'person'
};

var me = new person('Chuck', 'Tomasi');
var name = me.getDisplayName();
gs.info(name);
me.setFirstName('Fred');
me.setLastName('Luddy');
gs.info(me.getDisplayName());
```

For more information on script includes, watch:

TechNow episode on script includes

Lesson 34: Passing objects to functions

How to combine the power of objects with functions to go beyond basic data types being pass in and out of functions.

Link to lesson 34 video

Link to lesson 34 code repo

```
var item = Class.create();
item.prototype = {
    initialize: function() {
    },

    debugObject : function(obj) {

      gs.info('object=' + JSON.stringify(obj, null, 4));
    },

    type: 'item'
};

var myObj = {
  "type"   : "vehicle",
  "engine" : true,
  "wheels" : 4,
  "state"  : "allocated"
};
var myItem = new item();
myItem.debugObject(myObj);

// This is known as ‘loosely coupled’ – the object and the
// function are not dependent on each other.
```

Lesson 35: Class Inheritance

Creating classes based on existing classes to extend or override functionality, while avoiding duplicate code across classes.

Link to lesson 35 video

Link to lesson 35 code repo

```
var vehicle = Class.create();
vehicle.prototype = {
    initialize: function(year, make, model) {
      this.make = make;
      this.model = model;
      this.year = year;
    },

    register : function() {
      gs.info(this.getDisplayName() + ' registered!');
    },

    info : function() {
      gs.info('Vehicle info: TBD');
    },

    getDisplayName : function() {

      return this.year + ' ' + this.make + ' ' + this.model;

    },

    type: 'vehicle'
};

var car = Class.create();
car.prototype = Object.extendsObject(vehicle, {
```

```
    findDealer : function() {
      gs.info('Find dealer is not yet implemented');
    },

    info : function() {
      gs.info('Car info: TBD');
    },

    type: 'car'
});

var v1 = new vehicle('2018', 'John Deere', 'Tractor');
v1.register();
v1.info();

var v2 = new car('2017', 'Honda', 'CR-V');
v2.register();
v2.findDealer();
v2.info();
```

Lesson 36: Scripted REST APIs

This lesson builds on the skills you've learned thus far to create a custom REST API using a ServiceNow Scripted REST API. You'll learn about REST services and resources as well as how to pass information to your REST resource and return a result. This lesson also covers the basics of the REST API Explorer to help test your code.

Link to lesson 36 video

Link to lesson 36 code repo

System Web Services > Scripted Web Services

36.1 Simple scripted REST API (GET) resource

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

  return "hello, world!";

})(request, response);
```

36.2 Scripted REST API with query parameters

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

  // Example Query parameters
  // https://<instance_rest_endpoint>?active=true&name=now
  var queryParams   = request.queryParams;
  var isActiveQuery = queryParams.active; //true
  var nameQueryVal  = queryParams.name; //‘now’

  var answer = "Response: active=" + isActiveQuery + " name=" + nameQueryVal;

  return answer;
    })(request, response);
```

36.3 Scripted REST API with path parameters

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

  // Example path parameters
  // https://instance.service-now.com/api/now/myservice/{tableName}/{id}
  // https://instance.service-now.com/api/now/myservice/myApp_table/1234
  var pathParams = request.pathParams;
  var tableName  = pathParams.tableName; //‘myApp_table’
  var id         = pathParams.id; //‘1234’

  var answer = "Response: tableName=" + tableName + " id=" + id;

  return answer;

})(request, response);
```

36.4 Scripted REST API (POST) with request body payload

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var name  = request.body.data.name;
    var id    = request.body.data.id;
    var color = request.body.data.color;
    var answer = "Response: name=" + name + " id=" + id + " color=" + color;

    return answer;

})(request, response);
```

36.5 Scripted REST API with POST and response

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

    var name = request.body.data.name;
    var id    = request.body.data.id;
    var color = request.body.data.color;

    // Do some processing here

    var answer = {};
    answer.status = "OK";
    answer.author = "system";
    answer.item = {"name" : "Rome", "owner" : "Chuck Tomasi", "count" : 12};
    answer.active = true;
    response.setBody(answer);

    return response;

})(request, response);
```

How to create Scripted REST API in ServiceNow (SAASWITHSERVICENOW):

https://www.youtube.com/watch?v=48aBJqdC6xI

Lab 8a: Instructions

Rebuild the previous lab as a scripted REST API. Use Query parameters to accept the table name and limit.

Return the array of objects in the response body.

Get table record display values and sys_ids.

Lab 8a: Solution

Instructions:

Rebuild the previous lab as a scripted REST API. Use Query parameters to accept the table name and limit. Return the array of objects in the response body. Get table record display values and sys_ids.

Link to Lab 8a solution video

Solution:

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

    var queryParams = request.queryParams;
    var tableName   = queryParams.tableName;
    var limit       = queryParams.limit;

    var answer = [];
    var recGr = new GlideRecord(tableName);

    if (limit && limit > 0) {
        recGr.setLimit(limit);
    }
    recGr.query();

    while (recGr.next()) {
        var obj = {};
        obj.display_value = recGr.getDisplayValue();
        obj.sys_id = recGr.getUniqueValue();
        answer.push(obj);
    }
        response.setBody(answer);

})(request, response);
```

Lab 8b: Instructions

Rebuild the previous lab using a script include for the core logic. Use Query parameters to accept the table name and limit. Return the array of objects in the response body.

Lab 8b: Solution

Instructions:

Rebuild the previous lab using a script include for the core logic.

Use Query parameters to accept the table name and limit.

Return the array of objects in the response body.

Link to Lab 8b solution video

Solution:

Create a new script include (named 'SNJS'):

```
var SNJS = Class.create();
SNJS.prototype = {
    initialize: function() {
    },

    getRecords : function(tableName, limit) {

        var answer = [];
        var recGr = new GlideRecord(tableName);

           if (limit && limit > 0) {
            recGr.setLimit(limit);
        }
        recGr.query();

        while (recGr.next()) {
            var obj = {};
            obj.display_value = recGr.getDisplayValue();
            obj.sys_id = recGr.getUniqueValue();
            answer.push(obj);
        }
        return answer;
    },
    type: 'SNJS'
};
```

Test the script include using the following code:

```
var list = new SNJS().getRecords('problem', 5);
gs.info(list.length);                       // Returns the length of the list
gs.info(JSON.stringify(list, null, 4));     // Prints the list
```

Which could also be written as:

```
var sn = new SNJS();

var list = sn.getRecords('problem', 5);
gs.info(list.length);

gs.info(JSON.stringify(list, null, 4));
```

Now build the scripted REST API:

- Take the Record Finder REST API, add a new resource to it, called ‘Lab8b’
- Relative path = ‘/Lab8b’
- Add the following script:

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

    var queryParams = request.queryParams;
    var tableName   = queryParams.tableName;
    var limit.      = queryParams.limit;

    var list = new SNJS().getRecords(tableName, limit);

    response.setBody(list);

})(request, response);
```

Helpful Resources

This video provides some final notes and helpful resources to help you continue learning JavaScript on the Now Platform.

Link to this video

Link to list of resources

Resources:

- Codecademy.com (JavaScript)
- W3Schools.com (JavaScript)
- ServiceNow Developer Program
- Scripting in ServiceNow Fundamentals
- ServiceNow Docs
- ServiceNow Technical Best Practices
- TechNow webinars
- ServiceNow Community
- Stephen Bell's Scripting Best Practices Videos

There is also the integration series: https://devlink.sn/learn-integrations

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications.

https://code.visualstudio.com/

Using Visual Studio Code with ServiceNow

https://www.google.com/search?rlz=1C1CHZL_enNZ844NZ844&q=using+Visual+Studio+Code+with+servicenow&spell=1&sa=X&ved=2ahUKEwjt6aTvvJ3-AhXft1YBHYDvADkQkeECKAB6BAgIEAE

https://docs.servicenow.com/bundle/tokyo-application-development/page/build/applications/concept/vs-code.html