Introduction to Javascript

Information Visualization

Kalpathi Subramanian

Computer Science
The University of North Carolina at Charlotte

Last Modified: Jan18, 2023

References: *W3 Schools Link*, *The Modern Javascript*

What is Javascript?

- A functional scripting language developed for web based programming
- Works with HTML and can dynamically control HTML document content
- Syntax is largely similar to high level programming languages such as Java, C++ (Decision/Control, Loops)
- Implicitly typed, similar to other scripting languages.
- Highly flexible, needs carefully coding.
- **Relevance:** Basis for D3JS implements a Javascript API to visualization tools
- *W3 Schools Link* and *The Modern Javascript* provides excellent Javascript Documentation and Tutorials

What parts of Javascript should I know?

- Beyond the language basics (variables, assignments, control functions), most of the course assignments can be completed with Javascript functions; sufficient for event driven programming.
- Objects can be useful in more complex assignments; using simple dictionaries and classes is the most that might be required.
- Key points to note include passing/assigning variables is by **reference (similar to Java)**, types of variables must be carefully thought through, as Javascript makes conversions of types without error reporting. Beware!

Javascript Variables

const, let, var

- Declaring variables using *var* has **function** level scope
- Declaring variables using *let* has block level scope
- Declaring variables using **const** is used to set constant values, and cannot be reassigned.
- *W3 Schools Link* to more Details/Examples
- **Tip:** Always explicitly declare variables.

Javascript Syntax

Variables, Operators, Expressions, Keywords

- Syntax is similar to high level languages such as Java, C++ *W3 Schools Link*
- **Variables:** *var, let, const* *W3 Schools Link*
- **Operators:** Arithmetic, Assignment, String *W3 Schools Link*
- **Comments:** Single and Multi-line Comments, *W3 Schools Link*
- **Identifiers, Names:** Case sensitive

Javascript Control Functions: Conditional Statements

If/Else, Switch

- Supports *if/else* as in C++/Java *W3 Schools Link*

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

- Supports *switch* statement as in C++/Java; *switch* statement on string types supported. *W3 Schools Link*

```
switch(expression) {
  case x: // code block
    break;
  case y: // code block
    break;
  default: // code block
}
```

Javascript Control Functions: Loops

For Loops

- *for* loop syntax is identical to C++, Java. *W3 Schools Link*

```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

- Javascript also supports variants of for loops, as well as looping within data structures, such as arrays

While Loops

- *while* loop has a syntax that mirrors C++, Java. *W3 Schools Link*

```
while (condition) {
  // code block to be executed
}
```

- *continue*, *break* statements also supported to work with loops.

Javascript Functions

- Syntax:

```
function name(parameter1, parameter2,..., parameterN) {
  let v1, v2, ...vN;
  // code to be executed
}
```

- Uses the keyword *function* for definition.
- Similar to C++, Java, accepts arguments, can return values from the function using the keyword *return*
- *W3 Schools Link*

Function Expressions

In Javascript, function is a special kind of value, like a variable holding a value and used as such

```
function sayHi() {
  alert( "Hello" );
}
```

Function is a value

```
function sayHi() {   // create
  alert( "Hello" );
}
let func = sayHi;    // copy
func(); // Hello     // run the copy (it works)!
sayHi(); // Hello    // this still works too
alert(sayHi); // shows the function code
```

More info *The Modern Javascript*

Callback Functions

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
function showOk() {
  alert( "You␣agreed." );
}
function showCancel() {
  alert( "You␣canceled␣the␣execution." );
}
// usage: functions showOk, showCancel are passed as arg
ask("Do␣you␣agree?", showOk, showCancel);
```

Callback Functions

As Function Expressions

- Using anonymous functions..

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```

Javascript Arrays

Definition, Assignment, Copying

- Definition: *const animals = [cat, dog, tiger, bear, rhino];* *W3 Schools Link*
- Access, Assignment: By index.
- Copy Operation: Assigning to a new variable only assigns a reference.
- For a new copy, must create an Array object using the new operator, *const animals = new Array(cat, dog, tiger, bear, rhino)*
- Array Operations: A number of operations are defined on arrays: pop/push, length, concat, splice, slice *W3 Schools Link*

Javascript Objects

Properties, Methods

- Very similar to Java, C++ for defining objects

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
```

- Definitions: Can be thought of defining a dictionary *W3 Schools Link*
- Properties: Access to elements: person.firstName, person.fullName *W3 Schools Link*
- Similar to arrays, assigning an object copies its reference, not by value
- Must use new operator to create a fresh copy.
- Methods: *W3 Schools Link*

Javascript Math Library

- Use the **Math** library in Javascript
- Supports all needed mathematical functions
- Accessed as Math.SQRT(), Math.sin(), etc.
- Full list of Math functions are at *W3 Schools Link*

Asynchronous Executions

Call back functions Revisited (With Browser Methods[*The Modern Javascript*])

```
function loadScript(src) {
  // creates a <script> tag and append it to the page
  // this causes the script with given src to start loading and run when complete
  let script = document.createElement('script');
  script.src = src;
  document.head.append(script);
}
loadScript('/my/script.js');
// the code below loadScript
// doesn't wait for the script loading to finish
// ...
```

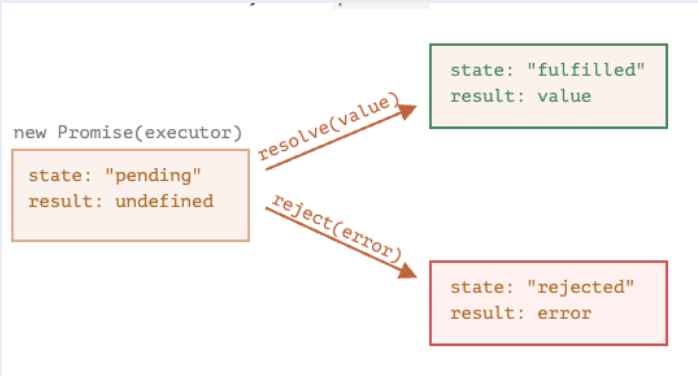
Use a callback to ensure before accessing functions in script

```
function loadScript(src, callback) {
  let script = document.createElement('script');
  script.src = src;
  script.onload = () => callback(script);
  document.head.append(script);
}
loadScript('https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js', script => {
  alert('Cool, the script is loaded');
  alert( _ ); // _ is a function declared in the loaded script
});
```

Promise

An elegant way to handle asynchronous execution

- Promise objects contain the **producer (executor)** code
- Has two functions, 'resolve', 'reject' as (function) arguments, one of which is executed after the executor completes.
- Promise objects must call resolve() or reject(), but not both!
- Only a single resolve() and reject() definition.
- reject() with error objects (inherit from Error)




```
let promise = new Promise(function(resolve, reject) {
// the function is executed automatically when the promise is constructed

// after 1 second signal that the job is done with the result "done"
setTimeout(() => resolve("done"), 1000);
});
```

Promise: Consumers: then, catch

then

- The 'then, catch' clauses receives the output (result or error) of the executor

```
let promise = new Promise(function(resolve, reject) {
   setTimeout(() => resolve("done!"), 1000);
});
// resolve runs the first function in .then
promise.then(
   result => alert(result), // shows "done!" after 1 second
   error => alert(error) // doesn't run
);
```

Or, in case of an error

```
let promise = new Promise(function(resolve, reject) {
   setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// reject runs the second function in .then
promise.then(
   result => alert(result), // doesn't run
   error => alert(error) // shows "Error: Whoops!" after 1 second
);
```

Additional Info at *The Modern Javascript*