The JavaScript Language

CS106E, Young

In this handout, we take a closer look at the JavaScript language. You may find the most efficient way to use this handout is to skim through it, getting a good sense of what's contained within, and then look things up here as needed as you write your JavaScript programs for our homework assignments.

As you'll discover JavaScript has a lot in common with PHP (just as this handout has much in common with the PHP Language Handout). Both languages are based on C. Both languages have untyped variables, parameters, and function return types. Both allow code outside of functions and classes.

If we start looking at a more advanced level, the languages are different. JavaScript objects are based on *prototypes* – an unusual model that currently is not used in any other programming language in widespread industry use – whereas PHP uses a traditional class model. However, at an introductory level, the languages are quite similar.

Variables

As I just mentioned, in JavaScript as with PHP, variables are untyped. The JavaScript rules for identifiers (which can be used for variables, function names, or object type names) are:

- They must begin with either a letter, underscore '_', or a dollar sign '$'.[1]
- After the initial letter or underscore, they may be followed by any combinations of letters, numbers, underscores, or dollar signs.

In contrast with PHP, however, JavaScript does not use dollar signs to denote variables. Here are some sample variable examples:[2]

```
x = 12.7;
salesTax = amount * 0.1;
```

JavaScript is case-sensitive. salesTax and SalesTax would be different variables – although you should never use two different identifiers that are the same, except for the case of their letters.

Variables can be declared in advance (but often aren't). As variables are untyped, use the keyword var for declaration:

```
var x;
var taxRate = 0.1;
```

---

[1] I don't recommend using the $ dollar sign though. Variables with dollar signs tend to get used by various libraries built on top of JavaScript to identify that they are special library variables.

[2] JavaScript prefers camel case for identifiers – taxRate is preferred although tax_rate would be legal. As described in the PHP handout, camel case refers to creating identifiers from multi-word strings, such as "Stanford computer science", by getting rid of the spaces and capitalizing all words past the first as in "stanfordComputerScience". It's called camel case as the capital letters form humps like a camel's back.

Data Types

JavaScript supports the following data types (in this class, we will not be covering the types written in italics):

Primitive Types

- Boolean
- Number
- String
- Undefined
- Null
- *Symbol*

Object Types

In addition, JavaScript supports Objects. There are a wide range of different types build using Objects including:

- Array
- Function
- String

You may have noticed I've listed String twice. There is a primitive String data type and a version built on top of JavaScript Objects. However, the two largely interoperate seamlessly, so you generally won't have to worry about differences between the two.

Boolean

The two boolean literals are true and false (always written in lower-case).

The number 0, the empty string "", NaN (not a number), null, and undefined all convert automatically to false. Everything else is considered true.

Boolean Operators

Boolean operators include:

&& — The double ampersand acts as a boolean "and":

```
if((age > 18) && ($age < 65)) { … }
```

|| — Two vertical bars '|' act as a boolean "or" operator.

```
discount = (age < 16) || (age >= 65);
```

! – The exclamation point acts as a boolean "not" operation. Example:

```
if(!(age >= 65)) { … }
```

Comparison Operators

Comparison operators include >, >=, <, <=.

The double equals "==" is used to compare if two values are the same. The exclamation point followed by an equals "!=" is used to see if two values are not the same. [3] For example:

```
if((university == "Stanford") { … }

if((university != "Stanford") { … }
```

Numbers

JavaScript only supports a single numeric data type. It does not distinguish between integers and floating-point numbers. It does support NaN for "Not a Number".

Arithmetic operators include:

+ for addition
- for subtraction
* for multiplication
/ for division
% for modulus

In addition, JavaScript supports C-style increment and decrement operators.

```
x++;
```

increments the value of the variable x by one and

```
x--;
```

decrements the value of the variable x by one.

Strings

JavaScript Strings can support Unicode, although the details are bit messy, so look it up if you plan to use anything other than the standard ASCII character set. JavaScript allows use of either single quotes or double quotes when writing string literals. In contrast to PHP, single-quoted strings and double-quoted strings work exactly the same. You can use the same escape sequences you're used to from C, C++, or Java, so a \t is a tab, a \n is a newline (e.g., a carriage return), a \\ give you a slash, a \' gives you a single quote and a \" gives you a double quote.

String Operators

JavaScript represents concatenating strings together using the + plus sign. This can cause some problems, as JavaScript commonly automatically converts from strings to numbers, but won't if you use the + sign. So in:

[3] You may also run into a triple equals "===". This operates similarly to "==" but does not allow type conversion. So if the two items being compared are of different types they are automatically not equals. There is a "!==" equivalent for not equals.

```
x = "5" - "3";
```

the strings will automatically convert to numbers and x will be the number 2. However,

```
y = "5" + "3";
```

won't result in 8, it will result in the string "53". This is why I used the parseFloat call in the Tax Example in the last handout.

Template Strings

In 2015, JavaScript added a new features called template strings. These allow us to perform variable substitution similar to what we saw with PHP strings. They also allow more complex embedded expressions. As of this writing, Template Strings are only supported by 91.5% of the web browsers in the US – most notably, it's not supported by Microsoft Internet Explorer, although it is supported by Microsoft's new Edge web browser.

You can create a template string by surrounding the string with back quotes `. The string can contain embedded expressions using a dollar sign followed by curly braces. Here are a few examples:

```
var cost = 12;
var amount = 100;
var amountSummary = `You purchased ${amount} items`;
var costSummary = `Your total cost is ${amount * cost}`;
```

Null and Undefined

JavaScript distinguishes between null (which it writes in all lower case) and undefined (also written in all lower case). undefined is the value of a variable that has not been assigned a value. null is a value that is used to indicate either that a variable normally would refer to an object, but currently does not, or that a function that normally should return an object is returning an exception condition instead (perhaps we're asking for the next item in a list, for example, and no such item exists).

Objects

Objects in JavaScript act as associative arrays.

Object Literals

We can create new objects on the fly by simply listing the properties and values we want to associate with the object.

```
var currStudent = {
    name: "Jane",
    year: "Senior"
  };

var dog = {
    name: "Molly",
    breed: "Terrier"
  };
```

Notice neither of these objects is based on a particular class. Nothing says that all dogs should have a name and breed or that students are limited to having a name and year. I could create another object representing a student skipping their year and writing:

```
var anotherStudent = {
    name: "Nicki",
    dorm: "Florence Moore Hall",
    state: "Calfornia"
  };
```

There are ways to create something similar to a traditional object-class structure. However, as we've just seen here, we can also create new objects on the fly, with no regard for classes.

Accessing Properties

We typically access properties using the dot notation like this:

```
var yearInSchool = currStudent.year;
```

However, using more traditional square brackets will also work:

```
var yearInSchool = currStudent["year"];
```

Note: *If you're not planning to write JavaScript code beyond what's needed for the HW, you can skip forward to the "Creating a New Object" subsection and then skip forward from there to the next major section on Arrays. Come back and read the rest of the Objects section, if you find yourself thinking that something seems very off with how JavaScript Objects work because somehow they act very differently from objects in other languages you've learned.*

Adding Methods

JavaScript supports functions as first-class objects.[4] A method in JavaScript is just a function that is a property of an object:

```
var rect = {
    width: 15,
    height: 10,
    area: function() {
       return this.width * this.height;
    }
  };
```

We can call the area function like this:

```
rect.area();
```

It's also theoretically possible to call it using the traditional square brackets:

```
rect["area"]();
```

Prototypes

JavaScript Objects aren't based on classes. Instead, they are based on a different object model called Prototypes. One object can be assigned another object as a prototype. Properties (including methods – which, as we've just seen, in JavaScript are just properties that have function objects assigned to them) can be inherited from their prototype.

This differs from traditional class-based systems in several respects:

[4] As we learned in the lecture on Programming Languages, in a language where functions are first class objects, functions can be created, assigned to variables, and passed into other functions as parameters.

- There is no distinction between classes and instances of classes. A prototype is an object, and is conceptually no different from the object that uses it as a prototype. In contrast, a class is conceptually very different from an object which is an instance of a class, and the two are not interchangeable.

- Because objects are not based on classes, there is no blueprint forcing objects based on a prototype to have a specific set of properties. Thus, two objects based on the same prototype can have different properties. This gives us a lot of flexibility, but can also cause problems if a programmer isn't careful.

  My example from the Object Literals section where one student had name and year properties and the other had name, dorm, and state properties would be impossible in a traditional class-based system. In class-based systems, all objects that are instances of a particular class must have the exact same properties.

The constructor is actually how objects get setup with shared prototypes. My object literals didn't get created by a constructor, so they don't have a prototype.[5] On the other hand, if I create two object using the Date constructor, they both share the Date prototype and thus share properties and methods.

Creating a New Object

As we've previously seen, we can create new object literals on the fly like this.

```
var firstFourth = {
          year: 1776,
          month: 7,
          day: 1
        };
```

However, in many cases we'll want to create a new object based on a pre-defined Object type. We do this by using a constructor for that Object type.

```
var moonLanding = new Date(1969,7,20);
```

With the object stored in the firstFourth variable, I can access or change the year, month, and day properties but that's pretty much it. With the object stored in moonLanding, I get all the properties and methods associated with the Date prototype:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Classes in JavaScript

Prototypes are actually a more powerful and more flexible system than classes. In 2015, traditional classes were added to JavaScript. A few things of note:

- As classes were not added until 20 years after the language was first introduced in 1995, the vast bulk of code out there is based on the concept of prototypes, not classes. The way JavaScript interacts with the web browser is fundamentally tied to the use of prototypes as well. So any use of JavaScript for web development depends on prototypes.

[5] Actually, there is an overall Object prototype that all objects inherit from.

- Classes are actually a syntactic sugar on top of the more powerful prototype model. When you define a class, internally it will be implemented using prototypes.

Arrays in JavaScript

As we've seen, JavaScript objects are essentially associative arrays. As with PHP, we can work with them as regular arrays though. In addition, there is an Array prototype that we can use that will give us access to different properties and methods especially useful for arrays.

Constructing an Array

We can create an array using the Array constructor – this version emphasizes that Array is just a type of object.

```
films = new Array(
                "Star Wars: The Last Jedi",
                "Beauty and the Beast",
                "The Fate of the Furious",
                "Despicable Me 3",
                "Jumanji: Welcome to the Jungle");
```

However, we can also use a square bracket notation that may feel more comfortable to many programmers:

```
films = [
            "Star Wars: The Last Jedi",
            "Beauty and the Beast",
            "The Fate of the Furious",
            "Despicable Me 3",
            "Jumanji: Welcome to the Jungle"
         ];
```

Iterating Through Arrays

We can iterate through Arrays using a standard for loop. Arrays have a length property that we can use to detect how many times we should perform our loop statements:

```
var outputHTML = "<ol>";

for(var i=0; i<films.length; i++) {
    outputHTML = outputHTML + "<li>" + films[i] + "</li>";
}

outputHTML = outputHTML + "</ol>";
```

There is also a special forEach method on an array. This method takes a function as a parameter and calls the function passed in on each item in the array. Here we go through the array and print out each element to the console log.

```
function logFilm(elem) {
    console.log(elem);
}

films.forEach(logFilm);
```

Control Structures

Control structures in JavaScript are very similar to those in C, C++, or Java.

Statements

JavaScript does not require semicolons at the end of statements, however it allows them as an option. I strongly recommend you put them there though, as getting in the habit of skipping semicolons will cause problems if you switch to most other languages.

Semicolons are not usually placed after blocks of statements which are enclosed in curly braces '{' '}'.

```
if (x > 5) {
  y = 3 * x;
  z = 3;
}
```

However, they are placed after curly braces used to indicate object literals.

```
var student = {
    name: "Jane",
    year: "Senior"
  };
```

If-Statements

If statements and if-else statements are very straightforward.

```
if(condition) {
  …statements…
}

if(condition) {
  …statements…
} else {
  …alternative-statements…
}
```

If you want to chain your if-else statements you use two separate words "else" and "if". The single word version found in PHP and some other languages is not available.

```
if(condition-1) {
  … statements-1 …
} else if(condition-2) {
  … statements-2 …
} else {
  … statements-3 …
}
```

While Loops

While loops are also very straightforward:

```
while(condition) {
  … statements …
}
```

For Loops

For loops are exactly the same as in C, C++, Java, and PHP. The for loop has a control section consisting of three parts, separated by semicolons. The first part sets the initial value for the index variable controlling the loop, the second is a test for stopping the loop, and the third increments the index variable:

```
for(init; test; increment) {
  … statements …
}
```

For example, here I have a loop which has a control variable i which goes from 0 to 9. Notice the test fails when i is 10. We increment by 1 each time.

```
for(var i = 0; i < 10 ; i++) {
  … statements …
}
```

Comments in JavaScript

JavaScript supports single line comments with a double slash //.

```
index = index + 2;  // we advance index by even numbers
```

Multi-line comments can be written using /* and */

```
/* This is a multline
   JavaScript comment */
```

Code Outside of Functions and Classes

Just as with PHP, JavaScript code does not need to be contained within a function or a class. I can simply put code between a script start and end tag:

```
<script>

var x = 10;

</script>
```

Functions

Declaration Format

As both JavaScript and PHP have untyped parameters and return values, JavaScript function declaration looks essentially the same as PHP function declaration:

```
function name(arg1, arg2, …, argN) {
   … statements …
}
```

Here's an example:

```
function interest(amount, rate, year) {
   return amount * Math.pow(1 + rate,year);
}
```

Note also the use of the keyword return to return a value from the function.

JavaScript does not use PHP's global keyword. You can simply access global variables directly.

```
rate = 0.05;

function interest(amount, year) {
   return amount * Math.pow(1 + rate, year);
}
```

Again, as mentioned in the PHP language handout, be cautious using global variables, they can make your code hard to follow and hard to maintain.

Learning More

You can learn more about JavaScript from several places:

https://developer.mozilla.org/en-US/docs/Web/JavaScript

https://www.w3schools.com/js/default.asp