LC 101

Unit 3 - JavaScript

February 27, 2017

JavaScript

- Included in all modern web browsers
  - Also recently seeing use outside of web browsers (i.e., node.js)
- Also called ECMAScript (after the association that is setting the standard)
  - Current browsers use ECMAScript 5

Comparison to Python (partial)

- Unlike Python, indentation does not matter
  - Python is perhaps the only programming language where indentation is syntactically important
  - Indentation is not *required* but should still be used for readability
- Curly braces {} are used to enclose blocks
  - **`if`**, **`while`**, and **`for`** blocks with only one line can skip the braces (though *I* prefer to always use them)
- **`if`**, **`while`**, and **`for`** conditional expressions must be enclosed in parentheses
- **`switch`** statement
- Logical **`and`**, **`or`**, and **`not`** are **`&&`**, **`||`**, and **`!`**

Comparison to Python (partial)

- Variables must be declared using **`var`**
- Functions are declared using **`function`**
- Two equals operators:
  - **`==`** does type coercion
  - **`===`** does not do type coercion (preferred unless type coercion is needed)
- **`undefined`** and **`null`** instead of None
  - **`undefined`** is for variables that are undeclared or declared and not assigned a value
  - **`null`** must be explicitly assigned to a variable
  - (I consider the existence of both of these to be a *design flaw* in the language)
- No classes (though JavaScript does have objects)

Comparison to Python (partial)

- Arrays in JavaScript are similar to lists in Python
  - Can change in size (unlike arrays in most other languages)
- (We will save objects for another time)
- No modules/import
  - Multiple JavaScript files can be loaded via an html file (or non-standard libraries)
- Comments
  - // for single line
  - /* */ for multi-line

Types

- Six basic types:
  - number (floats, ints, `Infinity`, `-Infinity`, `NaN`)
  - string (can use single or double quotes)
  - boolean (`true` and `false`)
  - object
  - function
  - undefined values (`undefined` and `null`)

Functions

- Two ways to define functions:

```
function add(x, y) {
  return x + y;
}

var add = function(x, y) {
  return x + y;
}
```

- Functions are *first-class* values
  - They can be assigned to variables and passed around as values

Scope

- Variables and functions are scoped to the function in which they are declared
  - Or globally if declared outside any function
- No block scope
  - `if`, `while`, etc., do not create new scopes
- Variable declarations, but not definitions, are moved to the top of their scope

```
function myFunction() {
  // ...some code...
  var x = 5;
  // ...more code...
}
```

is actually treated as

```
function myFunction() {
  var x;  // x is undefined
  // ...some code...
  x = 5;
  // ...more code...
}
```

Scope

- Function definitions using declaration notation, `function x() {...}`, are also moved to the top of their scope
- Functions defined using value notation, `var x = function() {...}`, are treated the same as other variables

Closures

- Functions returned from other functions retain access to the scope in which they were declared

```
function wrapValue(n) {
  var localVariable = n;
  return function() { return localVariable; };
}

var wrap1 = wrapValue(1);
var wrap2 = wrapValue(2);
console.log(wrap1());  // → 1
console.log(wrap2());  // → 2
```

Recursion

- A function can call itself
- A recursive function should have
  - a *base case* which ends the recursion
  - a *recursive call* which moves closer to the base case

```
function power(base, exponent) {
  if (exponent == 0)
    return 1;  // base case
  else
    return base * power(base, exponent - 1);  // recursive call
}

console.log(power(2, 3));  // → 8
```

Other

- JavaScript programs generally use camel-case instead of underscores
  - `myFunction` instead of `my_function`
  - not a requirement, just common practice
- `console.log()` is useful for development but generally should be removed for production code
  - Older versions of IE will throw an error on `console.log()` if the JavaScript console window is not actually open