Intro to JavaScript, by comparison to Java

Java	JavaScript
Developed at Sun Microsystems, since taken over by Oracle	Developed at Netscape, now maintained by Mozilla
Originally called Oak	Originally called LiveScript
Compiled language, bytecode interpreted by JVM	Scripting language, interpreted in browser
Object-oriented	Object-oriented
Strongly typed	Weakly typed
`int length = 12;` `String name = "Charles";` `int[] years = { 1881, 1991, 2002 };` `int year = years[2];` `int numYears = years.length;`	`var length = 12;` `var name = "Charles"; // or 'Charles'` `var years = [ 1881, 1991, 2002 ];` `var year = years[2];` `var numYears = years.length;`
Arithmetic operators: `+ - * / % ++ --` String operator: `+` Relational operators: `== != < <= > >=` Logical operators: `&& || !` Bit operators: `& | ^ ~ << >> >>>`	Arithmetic operators: same as Java String operator: same as Java Relational operators: all of Java's plus `=== !==` Logical operators: same as Java Bit operators: same as Java
`if ( boolean_expression ) {` `  // true part` `} else {` `  // false part` `}`	`if ( expression ) {` `  // true part` `} else {` `  // false part` `}` Note: the condition is not required to be Boolean. It will be considered false if it evaluates to `false`, `0`, `null`, the empty string, `NaN`, or undefined (var declared but no value). Otherwise it is true. That does *not* mean the false values are == to each other!
`for ( int i=0; i < 100; i++ ) {` `}`	`for ( var i=0; i < 100; i++ ) {` `}` Note: there is no exact equivalent to the Java "for-each" loop (e.g. `for (int yr : years)` where `years` is an array or collection of integers). All arrays however have a "forEach" method that can apply the supplied function to all elements of the array.
`while ( boolean_expression ) {` `}`	`while ( expression ) {` `}` Note: see IF for difference between `expression` and `boolean_expression`.

`` do { } while (boolean_expression); ```	``` do { } while (expression); ``` Note: see explanation of IF for difference between `expression` and `boolean_expression`.
`` switch (value) { case 1: break; case 2: break; default: } ``` Note: `value` is of an integer or (as of Java 7) String type.	``` switch (value) { case 1: break; case 2: break; default: } ```
`` System.out.println("hi"); ```	``` console.log("hi"); ```
Scope of variables ``` { int year = 2015; // more code } ``` Variable `year` is accessible only from the point of declaration to the end of the block in which it is declared. Except for non-private instance variables, that is.	Scope of variables ``` var global = 0; function func() { var local = 1; help = 2; } ``` Variable `global` is declared outside of any function so is considered global and accessible anywhere in the JavaScript program. This includes other JavaScript files if more than one is loaded. Variable `help` is not declared at all, but it will be declared implicitly as global even though it is first used inside a function. Dynamic scoping applies: `help` is global but cannot be referenced outside the function until the function is actually called. It is poor practice to not declare variables! Java does not have the concept of global variables, since all variables must be declared inside a class or method. Variable `local` is accessible only within `func`, the function in which it is declared. But it is accessible any time after defined until the function returns, even if it is defined inside of a nested block within the function. This is not the case if defined within an embedded function.

Action is specified by *methods* that are defined within classes. They are associated either with objects of that class (default) or with the class itself (static). `double convertCtoF(double c) {` `  return c * 1.8 + 32;` `}` `void exclaim(String s) {` `  String ex = s.toUpperCase() + "!";` `  System.out.println(ex);` `}`	Action is specified by *functions*. They can be standalone or associated with objects. The latter are called *methods*. The examples below are functions, not methods. Since Java does not have functions, these are not the exact equivalent of the Java code. But they give you a sense of the syntax. `function convertCtoF(c) {` `  return c * 1.8 + 32;` `}` `function exclaim(s) {` `  var ex = s.toUpperCase() + "!";` `  console.log(ex);` `}`
Java has anonymous *classes*, defined in-line. It did not have anonymous methods until Java 8. `this.addActionListener(new ActionListener(){` `  public void actionPerformed(ActionEvent e) {` `  System.out.println("Help me");` `  }` `});` Example of an anonymous class that implements the `ActionListener` interface.	Anonymous function. Function itself does not have a name but is assigned to a variable. The variable's name becomes the de facto function name for calling it later. `var message = function() {` `  alert("Help me");` `}` `message();`
Self-executing anonymous function Java does not have this since it is not a scripting language.	Self-executing anonymous function. A true anonymous function that will be invoked immediately instead of waiting to be called. `(function() {` `  alert("Help me");` `})();`

Defining a class, creating and using an object

```
class Temperature {
   private double temp = 0.0;
   public setTemp(double c) {
      this.temp = c;
   }
   public double getTemp() {
      return this.temp;
   }
   public double toF() {
      return this.temp * 1.8 + 32;
   }
}
```

(object is created and used in another class...)

```
Temperature t = new Temperature();
t.setTemp(10);
System.out.println(t.toF());
```

Defining and using an object

```
var t = {
   temp: 0,
   setTemp: function(c) {
      this.temp = c;
   },
   getTemp: function() {
      return this.temp;
   },
   toF: function() {
      return this.temp * 1.8 + 32;
   }
};

// use the methods
t.setTemp(10);
console.log(t.toF());
```

Alternative technique for defining this object (no Java equivalent)

```
var t = new Object();
t.temp = 0;
t.setTemp = function(c) {
   this.temp=c;
};
t.getTemp = function() {
   return this.temp;
};
t.toF = function() {
   return this.temp * 1.8 + 32;
};

// use the methods
t.setTemp(10);
console.log(t.toF());
```

Yes, this code creates an object that has no attributes, then the properties and methods are added dynamically! They can also be removed dynamically using the delete keyword, e.g. `delete t.getTemp;`

Defining a class, creating and using an object. Same as above, repeated here for side-by-side comparison.

```
class Temperature {
   private double temp = 0.0;
   public setTemp(double c) {
      this.temp = c;
   }
   public double getTemp() {
      return this.temp;
   }
   public double toF() {
      return this.temp * 1.8 + 32;
   }
}
```

(object is created and used in another class...)

```
Temperature t = new Temperature();
t.setTemp(10);
System.out.println(t.toF());
```

A technique for defining something akin to a class. The "constructor" function Temperature is the template

```
function Temperature() {
   this.temp = 0.0;
   this.setTemp = function(c) {
      this.temp = c;
   };
   this.getTemp = function() {
      return this.temp;
   };
   this.toF = function() {
      return this.temp * 1.8 + 32;
   };
}

// Create object and use methods
var t = new Temperature();
t.setTemp(10);
console.log(t.toF());
```

Yet another technique for defining something akin to a class. This is not in the Duckett book. It is a slight variation on the first technique.

```
var temperature = function() {
   var temp = 0.0;
   return {
      setTemp: function(c) {
         temp=c;
      },
      getTemp: function() {
         return temp;
      },
      toF: function() {
         return temp * 1.8 + 32;
      }
   };
};
// Create object and use methods
var t = new temperature();
t.setTemp(10);
console.log(t.toF());
```

Basic access to debugger and to web page HTML contents (DOM).

```
// Invoke browser's debugger now. Can place this anywhere
debugger;

// Get the element whose id attribute is "year"

var yearElem = document.getElementById("year");

// Get its contents (between the tags)

var year = yearElem.innerHTML;

year++;

// Modify its contents

yearElem.innerHTML = year;

// Write directly to the document.

document.write("Content courtesy of JavaScript");
```

`document` is a pre-defined object that refers to the internal HTML-element tree representing the current web page. This is called the Document Object Model (DOM). It has a lot of properties and methods, see p 126 for a few, chapter 5 for full coverage.

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

for extensive online documentation on JavaScript