Lab for JavaScript

Disclaimer: The information here may include errors, typos, or missing items. If so, notify to your instructor.

Practice #1 – Variable Scope
(No submission)

Scope of Variables in JavaScript, Courtesy of Triptych

```
var a = 1;

// global scope
function one() {
  alert(a);
}
```

```
var a = 1;

function two(a) {
  alert(a);
}

// local scope again
function three() {
  var a = 3;
  alert(a);
}
```

```
var a = 1;

function four() {
  if (true) {
    var a = 4;
  }

  alert(a); // alerts '4', not the global value of '1'
}
```

```
var a = 1;

function five() {
  this.a = 5;  // Object properties
}
```

```
var a = 1;

var six = (function() {
  var foo = 6;

  return function() {
    // JavaScript "closure" means I have access to foo in here,
    // because it is defined in the function in which I was defined.
    alert(foo);
  };
})();
```

```
var a = 1;

function seven() {
  this.a = 7;
}

// [object].prototype.property loses to
// [object].property in the lookup chain. For example...

// Won't get reached, because 'a' is set in the constructor above.
seven.prototype.a = -1;

// Will get reached, even though 'b' is NOT set in the constructor.
seven.prototype.b = 8;
```

```
// These will print 1-8
one();
two(2);
three();
four();
alert(new five().a);
six();
alert(new seven().a);
alert(new seven().b);
```

Practice #2 – Understanding Variable Life Time

(No submission)

Variable LifeTime, Credit: W3Schools.com

```
var counter = 0;

function add() {
    counter += 1;
}

add();
add();
add();

// the counter is now equal to 3
```

```
var counter = 0;

function add() {
    counter += 1;
}

add();
add();
add();

// the counter is now equal to 3
```

```
function add() {
    var counter = 0;
    counter += 1;
}

add();
add();
add();

// the counter should now be 3, but it does not work !
```

```
function add() {
    var counter = 0;
    function plus() {counter += 1;}
    plus();
    return counter;
}
```

```
var add = (function () {
    var counter = 0;
    return function () {return counter += 1;}
})();

add();
```

```
add();
add();

// the counter is now 3
```

The variable **add** is assigned the return value of a self invoking function. The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression. This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope. This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables. The counter is protected by the scope of the anonymous function, and can only be changed using the add function.