Closures

Closure

In JavaScript, a closure is any function that keeps reference to variables from its parent's scope *even after the parent has returned*. This means practically any function can be considered a closure, because, as we learned in the variable scope a function can refer to, or have access to –

- any variables and parameters in its own function scope
- any variables and parameters of outer (parent) functions
- any variables from the global scope.

But our aim is not just to use them – it is to understand them.

If we don't understand how they work, we can't use them *properly*. For that reason, we are going to split the above closure definition into three easy-to-comprehend points

Point 1: *You can refer to variables defined outside of the current function.*

```
function setLocation(city) {
  var country = "France";

  function printLocation() {
    console.log("You are in " + city + ", " + country);
  }

  printLocation();
}

setLocation ("Paris");  // output: You are in Paris, France
```

Point 2: *Inner functions can refer to variables defined in outer functions even after the latter have returned.*

```
function setLocation(city) {
  var country = "France";

  function printLocation() {
    console.log("You are in " + city + ", " + country);
  }

  return printLocation;
}

var currentLocation = setLocation ("Paris");

currentLocation();   // output: You are in Paris, France
```

Point 3: *Inner functions store their outer function's variables by reference, not by value.*

```
function cityLocation() {
  var city = "Paris";

  return {
    get: function() { console.log(city); },
    set: function(newCity) { city = newCity; }
  };
}

var myLocation = cityLocation();

myLocation.get();               // output: Paris
myLocation.set('Sydney');
myLocation.get();               // output: Sydney
```

- Closures can both read and update their stored variables,
- The updates are visible to any closures that have access to them.
- Closures store *references* to their outer variables, rather than copying their values.
- A very important point to remember, because not knowing it can lead to some hard-to-spot logic errors – as we saw in the “Immediately-Invoked Function Expressions (IIFEs)”.