Interview Cake

← course home (/table-of-contents)

Closures (In JavaScript and Beyond)

A closure is a function that accesses a variable "outside" itself. For example:

JavaScript

```
var message = 'The British are coming.';
function sayMessage(){
    alert(message);  // here we have access to message,
    // even though it's declared outside this function!
}
```

We'd say that `message` is "closed over" by `sayMessage()`.

One useful thing to do with a closure is to create something like an "instance variable" that can change over time and can affect the behavior of a function.

JavaScript

```
// function for getting the id of a dom element,
// giving it a new, unique id if it doesn't have an id yet
var getUniqueId = (function(){
    var nextGeneratedId = 0;
    return function(element) {
        if (!element.id) {
            element.id = 'generated-uid-' + nextGeneratedId;
            nextGeneratedId++;
        }
        return element.id;
    };
})();
```

Why did we put `nextGeneratedId` in an immediately-executed anonymous function? It makes `nextGeneratedId` private, which prevents accidental changes from the outside world:

JavaScript

```
// function for getting the id of a dom element,
// giving it a new, unique id if it doesn't have an id yet
var nextGeneratedId = 0;
var getUniqueId = function(element) {
    if (!element.id) {
        element.id = 'generated-uid-' + nextGeneratedId;
        nextGeneratedId++;
    }
    return element.id;
};

// ...
// somewhere else in the codebase...
// ...

// WHOOPS--FORGOT I WAS ALREADY USING THIS FOR SOMETHING
nextGeneratedId = 0;
```

⬅ course home (/table-of-contents)

Next up: In-Place Algorithms ➔ (/concept/in-place?section=javascript&course=fc1)