Closures (In JavaScript and Beyond)

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

```JavaScript
const 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
const getUniqueId = (() => {
  let nextGeneratedId = 0;
  return 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
let nextGeneratedId = 0;
const getUniqueId = 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;
```

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