CS454 Node.js & Angular.js

Cydney Auman
CSULA

More Topics In Javascript - Week 2

Scope

Global Scope is a variable declared outside a function definition. And its value is accessible and modifiable throughout your program.

Local Scope a variable that is declared inside a function definition is local. It is created and destroyed every time the function is executed, and it cannot be accessed by any code outside the function

```
var x = "global";

function myFunction() {
    var y = "local";

};
```

Context

Context refers to the value of the **`this`** keyword.

Inside a function, the value of **`this`** depends on how the function is called. When code is not in strict mode, the value of **`this`** must always be an object so it defaults to the global object.

When a function is called as a method of an object, its **`this`** is set to the object the method is called on.

Closure

In JavaScript, an inner (nested) function stores references to the local variables that are present in the same scope as the function itself, even after the function returns. This set of references is called a **closure.**

```
function greet(name) {
  // Local variable 'name' is stored in the closure for the inner function.
  return function() {
    console.log('Hello ' + name);
  }
}

var sendGreeting = greet('Cydney');
sendGreeting();
```

Object Prototyping

In a traditional class-based language (such as Java or C++), to create an object you must first create a class which defines that objects properties and methods. The class is used as a blueprint to know how to construct the object, just in a similar manner to how a house is constructed by following an actual blueprint.

In JavaScript, there are no classes, only objects and primitive data types. Really for the sake of simplicity - everything in JavaScript is an object, from numbers, to strings, and even functions.

Object Prototyping

Prototype-based programming is a style of object-oriented programming in which classes are not present, and behavior reuse (known as inheritance) is accomplished through a process of adding methods and properties.

All JavaScript objects inherit their properties and methods from their prototype.

Callbacks

Simply put a **callback** is a function to be executed after another function is executed.

Callbacks in JavaScript are functions that are passed as arguments to other functions. This is a very important feature of asynchronous programming, and it enables the function that receives the callback to call our code when it finishes a long running task, while allowing us to continue the execution of other code.

Callbacks

```
function genRandom(n1, n2, callback) {
  var myNumber = Math.floor(Math.random() * n1) + n2;
  callback(myNumber);
}


genRandom(10, 1, function done(result) {
  console.log(result);
})
```

Callbacks & the Event Loop

The **event loop** is a programming construct that waits for and dispatches events/messages in a program.

In Javascript The **event loops** job is to look at the stack and look at the task queue. If the stack is empty it takes the first thing on the queue and pushes it on to the stack which effectively run it.

```
console.log("Hello!");

setTimeout(function () {
  console.log("setTimeout is asynchronous!");
}, 3000);

console.log("JavaScript is largely synchronous");
```

Garbage Collection

Garbage collection is a form of memory management. It’s where we have the notion of a collector which attempts to reclaim memory occupied by objects that are no longer being used. In a garbage-collected language such as JavaScript, objects that are still referenced by your application are not cleaned up.

Garbage Collection - Fun Facts

Should you de-reference objects?

Manually de-referencing objects is not necessary in most cases. By simply putting the variables where they need to be (ideally, as local as possible, i.e. inside the function where they are used versus an outer scope), things should just work.

Can you force Garbage Collection in JS?

It’s not possible to force garbage collection in JavaScript. You wouldn’t want to do this, because the garbage collection process is controlled by the runtime, and it generally knows best when things should be cleaned up.

What Fundamental Problem Does Garbage Collection Solve?

The fundamental problem garbage collection solves is to identify dead memory regions (unreachable objects/garbage) which are not reachable through some chain of pointers from an object which is live. Once identified, these regions can be re-used for new allocations or released back to the operating system