Welcome to JavaScript Functions series. This is the third article in the series focusing on **Anonymous Functions**. In the earlier articles, we covered JavaScript Function Declaration and Function Expression and its pros and cons. We will continue to follow similar pattern in this article.

**Anonymous Functions**

As the name suggests anonymous functions are functions without any given name. Now, you might be thinking how is it even possible and most importantly useful? If we define a function without any name, how can we even call it? Well, in large scale JavaScript programs anonymous functions plays important role, however we will see there are some gotchas associated with it and it is not always recommended to use anonymous functions. So, let’s first understand how to declare these types of functions.

JavaScript provides different ways to declare an anonymous function in a program. We will cover commonly used options in this article.

**Anonymous Functions Declaration**

Anonymous functions can be declared similar as function expression syntax, except that the function is declared without any name. In either case, once you assign a function to a variable [log as shown in below example], you have to use the assigned variable to call the function.

var log = function (message){

console.log(message);

}

log("Jack & Jill went up the hill…");

Hoisting rules for function expression and anonymous functions are same. That means; JavaScript engine hoist the function variable instead of complete function definition. So JavaScript runtime re-implements the above code as -

var log;

log = function(message){

console.log(message);

}

log("Jack & Jill went up the hill…");

Note again, it doesn’t hoist function definition completely. It just hoists the function variable and initializes it where function was initially defined. So in this case, you cannot call the function unless it is explicitly defined earlier. Doing so will result into a runtime exception undefined is not a function since we are trying to call a function which has not been defined.

var log;

log("JavaScript is my favorite language"); // throws exception

log = function (message){

console.log(message);

}

Another way of declaring an anonymous function is within object literal. It is common practice, especially when you want to hide implementation details from the end user. In below defined code listing, we have an anonymous function getFullName which concatenates the firstName and lastName values and return it to caller.

var person = {

firstName : “Prasad”,

lastName : “Honrao”,

getFullName: function(){

return this.firstName + “ “ + this.lastName;

}

};

console.log(person.getFullName());

There are various JavaScript design patterns associated with defining anonymous functions in object literal like module pattern, revealing module. The purpose of these patterns is to hide complex implementation detail from the caller. I will try to cover these patterns in subsequent articles; however you should be able to find various articles on web explaining these patterns.

Now let’s take a look at some of the JavaScript framework and libraries and understand how anonymous functions are used in different scenarios.

**Anonymous Functions As Function Object**

JavaScript allows you to pass function as an object to other functions. You don’t need to explicitly define the function before passing it as a parameter to another function, rather you can utilize anonymous function feature as shown in below example.

Our anonymous function is passed to setTimeout, which will execute the function in 1000 milliseconds and will log hello world to console.

setTimeout(function() {

console.log('hello world');

}, 1000);

It’s not mandatory to pass only anonymous function to setTimeout method as shown earlier. You can easily rewrite code using function expression syntax as shown below.

var say = function(){

console.log("Hello M");

}

setTimeout(say, 1000);

I personally prefer declaring the functions upfront and then passing it as an argument to other functions, as it makes code more readable and easier to debug.

**Anonymous Functions As Callback Functions**

Anonymous functions are commonly used as callback functions in many JavaScript frameworks and libraries.

Popular JavaScript framework **Node.js** heavily uses anonymous functions feature of the language. Below code snippet creates a simple HTTP server using http module’s createServer method.

var http = require("http");

var server = http.createServer(function(request, response) {

response.write("Hello World");

response.end();

});

server.listen(1234);

The createServer method accepts a function parameter which gets executed for each HTTP request. In above case, we have passed an anonymous function, which gets executed for each HTTP request.

Another useful and widely used JavaScript library jQuery uses anonymous function feature in almost all of its APIs. Below code snippet display hello message to the user once he clicks on helloButton.

// event handler

$("#helloButton").click = function(){

alert("hello");

};

Declaring an anonymous function as callback function in our own program is simple process. In below code example, we have defined an object *Contest* which contains a function *askQuestion* which takes function as argument and calls it during its execution. During *askQuestion* function execution, we can simply pass an anonymous function as an argument, which then gets execute as callback function and returns the result to the caller.

var Contest = {

ans: "Red",

askQuestion: function (answered) {

console.log("Your fav color :" + this.ans);

answered(this.ans); // callback function

}

}

Contest.askQuestion(function (answer) {

console.log("Answer is " + answer);

});

**Disadvantages**

We have seen that anonymous functions are powerful feature of JavaScript language; there are few disadvantages of using it in large scale apps.

1. Debugging anonymous functions is always a tedious problem as you don’t get much information in call stack, since these are unnamed functions.
2. These functions cannot be unit tested easily as they are passed as argument to an external function. Most of the times you need to refactor the code to improve unit testing and code coverage of your application.
3. As these functions are nested inside other functions, it impacts the code readability and structuring. You need to provide proper code comments so that other members in the team can understand the function clearly.
4. These functions cannot be reused.