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

**Anonymous Functions Declared As Function Expression**

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);

}

**Anonymous Functions Declared In Object Literal**

Declaring an anonymous function in an object literal is common practice, especially when you’re designing an API for client application to consume. Anonymous functions are useful to hide implementation details

var person = {

firstName : “Prasad”,

lastName : “Honrao”,

fullName: function(){

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

}

};

console.log(person.fullName());

**Anonymous Functions Declared In Node.js**

var http = require("http");

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

response.write("Hello World");

response.end();

});

server.listen(1234);

We define the function we want to pass to execute right there at the place where execute expects its first parameter. This way, we don’t even need to give the function a name, which is why this is called an anonymous function.

**Anonymous Functions Declared With jQuery**

// event handler

$("p").click = function(){

alert("hello");

};

**Anonymous Functions As Function Object**

setTimeout(function() {

alert('hello');

}, 1000);

// Our anonymous function is passed to setTimeout, which will execute

// the function in 1000 milliseconds.

**Anonymous Functions As Callback Functions**

Anonymous functions are commonly used as callback functions. In below code example, we have defined an object literal *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 returns the answer to the console.

var Contest = {

ans: "Red",

askQuestion: function (answered) {

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

answered(this.ans);

}

}

Contest.askQuestion(function (answer) {

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

});

**No Readability Traps**

Let’s take a look at readability trap we discussed in function declaration example in earlier post and redefine *print* functions using anonymous function syntax.

var print = function (input) {

console.log("Print was called with value " + input);

}

print(10);

var print = function (input) {

console.log("Redefining print with value " + input);

}

print(20);

In this case, after calling *print* function *print(10)*, we are overriding *print* function definition, hence both the calls to *print* function will display output as shown below.

C:\$\Code - Git\Blog\2014\08\JavaScript Functions Part 1 - Function Declaration\assets\Output1.PNG

**Limitations**