Immediately Invoked Function Expressions

IIFE (pronounced “Iffy”) is an abbreviation for Immediately Invoked Function Expression, and the syntax looks like this:

(function () {

// Do fun stuff​

}

)()

It is an anonymous function expression that is immediately invoked, and it has some particularly important uses in JavaScript.

# How Immediately Invoked Function Expressions Work?

* The pair of parenthesis surrounding the anonymous function turns the anonymous function into a function expression or variable expression. So instead of a simple anonymous function in the global scope, or wherever it was defined, we now have an unnamed function expression.
* It is as if we have this:

*// Shown without the parentheses here:​*

? = function () {};

*​// And with the parentheses here:​*

(? = function () {});

*​// An unknown variable assigned the value of a function, wrapped in a parenthesis, which turns it into an unnamed function expression.*

* Similarly, we can even create a **named**, immediately invoked function expression:

(showName = function (name) {

console.log(name || "No Name")

}) ();

// No Name​

showName ("Rich"); // Rich​

showName (); // No Name

## Note

* that you cannot use the var keyword inside the opening pair of parentheses (you will get a syntax error), but it is not necessary in this context to use var since any variable declared without the var keyword will be a global variable anyway.
* We were able to call this named function expression both immediately and later because it has a name.
* But we can’t call the anonymous function expression later, since there is no way to refer to it. This is the reason it is only useful when it is immediately invoked.
* By placing the anonymous function in parentheses (a group context), the entire group is evaluated, and the value returned. The returned value is the entire anonymous function itself, so all we must do is add two parentheses after it to invoke it.
* Therefore, the last two parentheses tell the JS compiler to invoke (call) this anonymous function immediately, hence the name “**Immediately Invoked Function Expression**.”
* Because JavaScript has function-level scope, all the variables declared in this anonymous function are local variables and therefore cannot be accessed outside the anonymous function.
* So we now have a powerful piece of anonymous code inside an unnamed function expression, and the code is meaningless unless we invoke the anonymous function, because nothing can access the code. It is the immediate invocation of the anonymous function that makes it powerful and useful.
* You can pass parameters to the anonymous function, just like you would any function, including variables. The anonymous function’s scope extends into any outer function that contains it and to the global scope.

# When To Use Immediately Invoked Function Expressions?

## To Avoid Polluting the Global Scope

The most popular use of the IIFE is to avoid declaring variables in the global scope. Many JavaScript libraries use this technique, and of course many JS pros, too. It is especially popular amongst jQuery plugin developers. And you should use an IIFE in the top-level (main.js) of your applications.  
  
In this first example, we are using it in the global scope to keep all our variables local to the anonymous function, and thus outside the global scope where variables can shadow (block) other already-defined variables with the same name (probably from an included library or framework). All of our code for the application will start in the IIFE:

*// All the code is wrapped in the IIFE​*

(function () {

​ var firstName = “Richard”;

function init () {

doStuff (firstName);

*// code to start the application​*

​ function doStuff () {

*// Do stuff here​*

}

​function doMoreStuff () {

*// Do more stuff here​*

}

*​ // Start the application*

init ();

}) ();

* Note that you can also pass jQuery or any other object or variable via the parameter (the last 2 parentheses).

## Use With the Conditional Operator

The use of the IIFE in this manner is not as well known, but it quite powerful since you can execute complex logic without having to setup and call a named function:

var unnamedDocs = [], namedDocs = ["a\_bridge\_runover",   
 "great\_dreamers"];

​function createDoc(documentTitle) {

var documentName = documentTitle ?

​(function (theName) {

var newNamedDoc = theName.toLocaleLowerCase().replace(" ", "\_");

namedDocs.push(newNamedDoc);

return newNamedDoc;

})(documentTitle)

:

(function () {

var newUnnamedDoc = "untitled\_" + Number(namedDocs.length + 1);

unnamedDocs.push(newUnnamedDoc);

return newUnnamedDoc;

})();

​return documentName;

}

​createDoc("Over The Rainbow"); // over\_the rainbow​

​createDoc(); // untitled\_4

## Use it in Closures to Prevent Fold Over

To prevent close over in for loops, we can use an Immediately Invoked Function Expression to prevent a common bug when closures are used with for loops:

To fix side effects (bug) in closures, you can use an IIFE, such as if this example:

function celebrityIDCreator (theCelebrities) {

var i;

var uniqueID = 100;

for (i = 0; i < theCelebrities.length; i++) {

theCelebrities[i]["id"] = function (j) {

*// the j parametric variable is the i passed in on invocation of this IIFE​*

return function () {

return uniqueID + j;

*// each iteration of the for loop passes the current value of i into this IIFE and it saves the correct value to the array​*

}

} (i);

*// immediately invoke the function passing the i variable as a parameter​*

}

return theCelebrities;

}

​var actionCelebs = [

{name:"Stallone", id:0},

{name:"Cruise", id:0},

{name:"Willis", id:0}

];

​var createIdForActionCelebs = celebrityIDCreator (actionCelebs);

​var stalloneID = createIdForActionCelebs [0];

console.log(stalloneID.id()); // 100​

​var cruiseID = createIdForActionCelebs [1];

console.log(cruiseID.id()); // 101