# JAVASCRIPT SYNTAX

**Traditional Way**:

function *functionName*(*parameters*) {
  *code to be executed*
}

| Type | Example | Explanation |
|---|---|---|
| Variable Assignment | // Named<br>var sum = function add(num1,num2) {<br>  return num1 + num2;<br>};<br><br>// Anonymous<br>var sum = function(num1, num2) {<br>  return num1 + num2;<br>}; | This is functionally equivalent to the Function Declaration above, except this variation is NOT hoisted. The name of our function (add) cannot be called directly (see below), but it can come in handy when debugging:<br><br>sum(1,1);<br>// returns 2<br><br>add(1,1);<br>// "add is not defined" |
| Immediately invoked | // Named<br>(function sum(num1, num2) {<br>  return num1 + num2;<br>}(1, 2));<br><br>// Anonymous<br>(function(num1, num2) {<br>  return num1 + num2;<br>}(1, 2)); | This function is immediately invoked, meaning that it is defined and called at the same time. The function's name is only available within its execution scope (defined by the parentheses), so it cannot be called later in the program.<br>sum(1,1);<br>// "sum is not defined"<br>Immediately invoked functions can be used to encapsulate a program, preventing it from polluting the global namespace. |
| Assigned and Invoked | // Named<br>var sum = function add(num1, num2) {<br>  return num1 + num2;<br>}(1, 2);<br><br>// Anonymous<br>var sum = function(num1, num2) { | This is a combination of the variable assignment expression and the immediately invoked function (both demonstrated above). One neat application for the named variety of this is to make recursive functions more readable, by substituting arguments.callee with your function name. |

| Type | Example | Explanation |
|---|---|---|
| | return num1 + num2;<br>}(1, 2); | |
| Property Assignment | // Named<br>var obj1 = {<br>sum: function add(num1, num2) {<br>return num1 + num2<br>}<br>};<br><br>// Anonymous<br>var obj2 = {<br>sum: function(num1, num2) {<br>return num1 + num2<br>}<br>}; | By assigning functions (either named or unnamed) to properties of objects, we define methods on those objects. This has many applications in object oriented programming. We can also use this to namespace our functions, and keep them out of the global scope.<br><br>Here's how we would call the methods defined in the examples:<br><br>obj1.sum(1, 2);<br>// returns 3<br><br>obj2.sum(1, 2);<br>// returns 3 |
| Passed as Argument | // Named<br>window.setTimeout(function add() {<br>alert(1 + 2);<br>}, 500);<br><br>// Anonymous<br>window.setTimeout(function() {<br>alert(1 + 2);<br>} 500); | Function names in ECMAScript are nothing more than variables, meaning we can pass them around like variables. Many methods (like setTimeout()) take functions as arguments. This is a common pattern for defining callbacks.<br><br>In these examples, we define them in line, but we could also pass in predefined functions. |
| Returned (closure) | // Named<br>function counter() {<br>var count = 0;<br>**return function c() {**<br>**alert(count++);**<br>**}**<br>}<br><br>// Anonymous<br>function counter() { | Functions can be returned from other functions. This lets us do clever things like the following (picking up from the example):<br><br>var bob = {}, rob = {};<br>bob.count = counter();<br>rob.count = counter();<br>bob.count(); // alerts "0"<br>bob.count(); // alerts "1" |

| Type | Example | Explanation |
| --- | --- | --- |
| | var count = 0;<br>**return function() {**<br>**alert(count++);**<br>**}**<br>} | rob.count(); // alerts "0"<br>rob.count(); // alerts "1"<br><br>Each person can increment their own count variable despite the fact that it originated outside the scope of the count() function. This is called a closure |
| Arrow Functions | // There is no "named"<br>// form of this function.<br><br>// Anonymous<br>var sum = (num1, num2) => {<br>return num1 + num2<br>};<br><br>// Anonymous w/out optional<br>// bracketed return<br>var sum = (num1, num2) =><br>num1 + num2; | Arrow (or "fat-arrow") functions are newly introduced as part of the ES6 specification. As such, they are only implemented in the most modern browsers (check a compatibility table for specifics). This example:<br><br>var sum = (num1, num2) => {<br>return num1 + num2<br>};<br><br>is functionally equivalent to:<br><br>var sum = function(num1, num2) {<br>return num1 + num2;<br>};<br><br>Arrow functions are function expressions, and as such, they are usable in all the contexts shown above (variable assignment, passed as an argument, etc.). You may see slight syntactical variations across uses (as shown with in the example with optional brackets |