Function Name	Function Description	Example
Function	A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).	NA

Function Syntax	A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
	
	Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
	
	The parentheses may include parameter names separated by commas:
	(parameter1, parameter2, ...)
	
	The code to be executed, by the function, is placed inside curly brackets: {}
	
	Function parameters are listed inside the parentheses () in the function definition.
	
	Function arguments are the values received by the function when it is invoked.
	
	Inside the function, the arguments (the parameters) behave as local variables.
		// Function to compute the product of p1 and p2
		
		function myFunction(p1, p2) {
		  return p1 * p2;
		}

Function Invocation	The code inside the function will execute when "something" **invokes** (calls) the function: When an event occurs (when a user clicks a button) When it is invoked (called) from JavaScript code Automatically (self invoked)	NA

Function Return	When JavaScript reaches a `return` statement, the function will stop executing.
	
	If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
	
	Functions often compute a return value. The return value is "returned" back to the "caller".
		// Function is called, the return value will end up in x
		let x = myFunction(4, 3);
		
		function myFunction(a, b) {
		// Function returns the product of a and b
		  return a * b;
		}

The () Operator	The () operator invokes (calls) the function.
	
	Accessing a function with incorrect parameters can return an incorrect answer.
	
	Accessing a function without () returns the function and not the function result.
	
	As you see from the examples above, toCelsius refers to the function object, and toCelsius() refers to the function result.
		Eg 1: function toCelsius(fahrenheit) {
		return (5/9) * (fahrenheit-32);
		}
		
		let value = toCelsius(77);
		
		Eg 2: function toCelsius(fahrenheit) {
		return (5/9) * (fahrenheit-32);
		}
		
		let value = toCelsius();
		
		Eg 3: function toCelsius(fahrenheit) {
		return (5/9) * (fahrenheit-32);
		}
		
		let value = toCelsius;

Function Expressions	A JavaScript function can also be defined using an expression. A function expression can be stored in a variable. After a function expression has been stored in a variable, the variable can be used as a function. The function above is actually an anonymous function (a function without a name). Functions stored in variables do not need function names. They are always invoked (called) using the variable name. The function above ends with a semicolon because it is a part of an executable statement.	const x = function (a, b) {return a * b}; let z = x(4, 3);

Function Hoisting	Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope.
	Hoisting applies to variable declarations and to function declarations.
	Because of this, JavaScript functions can be called before they are declared:
	Functions defined using an expression are not hoisted.
		myFunction(5);
		
		function myFunction(y) {
		  return y * y;
		}

Self-Invoking Functions	Function expressions can be made "self-invoking".
	A self-invoking expression is invoked (started) automatically, without being called.
	Function expressions will execute automatically if the expression is followed by ().
	You cannot self-invoke a function declaration.
	You have to add parentheses around the function to indicate that it is a function expression
		(function () {
		  let x = "Hello!!"; // I will invoke myself
		})();

Arrow Functions	Arrow functions allows a short syntax for writing function expressions. You don't need the function keyword, the return keyword, and the curly brackets. Arrow functions do not have their own this. They are not well suited for defining object methods. Arrow functions are not hoisted. They must be defined before they are used. Using const is safer than using var, because a function expression is always constant value. You can only omit the return keyword and the curly brackets if the function is a single statement. Because of this, it might be a good habit to always keep them	const x = (x, y) => { return x * y };