**Functions**

**Introduction to functions**

A ***function*** is a named group of statements. JavaScript functions are declared with the function keyword followed by the function name and parameter list in parentheses (). A ***parameter*** is a variable that supplies the function with input. The function's statements are enclosed in braces {}.

Invoking a function's name, known as a ***function call***, causes the function's statements to execute. An ***argument*** is a value provided to a function's parameter during a function call.

Construct 6.6.1: Function declaration.

function functionName(parameter1, parameter2, ...) {

*// Statements to execute when function is called*

}

Declaring and Calling Functions :

Graphical user interface, text, application

Description automatically generated

Good practice is to use function names that contain a verb and noun. Ex: display is a vague function name, but displayAverage is better because displayAverage indicates what is being displayed.

Good practice is to use camel case for JavaScript function names, where the name starts with a lowercase letter and subsequent words begin with a capital letter.

**Returning a value**

A function may return a single value using a ***return*** statement. A function that is missing a return statement returns undefined.

Graphical user interface, application

Description automatically generated

1. A function named findAverage is declared with two parameters: num1 and num2.
2. findAverage() is called with arguments 6 and 7, which are assigned to parameters num1 and num2.
3. The return statement returns the average of num1 and num2, which is 6.5.

**Function expressions and anonymous functions**

JavaScript functions may be assigned to a variable with a function expression. A ***function expression*** is identical to a function declaration, except the function name may be omitted. A function without a name is called an ***anonymous function***. Anonymous functions are often used with arrays and event handlers, discussed elsewhere in this material.

Figure Assigning a function expression to a variable.

*// Function name is omitted*

let displaySum = function(x, y, z) {

console.log(x + y + z);

}

*// Function call*

displaySum(2, 5, 3);

Unlike functions declared with a function declaration, a variable assigned with a function expression cannot be used until after the variable is assigned. Using a variable before the variable is assigned with a function expression causes an exception.

Using a function expression before assignment.

Graphical user interface, text, application

Description automatically generated

1. findLargest() may be called before the findLargest() function declaration.
2. Since x > y, findLargest() returns 5, and 5 is output to the console.
3. Calling displaySum() before displaySum is assigned with a function expression produces an exception.

Scope and the global object

**The var keyword and scope**

1. In addition to declaring variables with let, a variable can be declared with the ***var*** keyword. Ex: var x = 6; declares the variable x with an initial value of 6. When JavaScript was first created, var was the only way to declare a variable. The let keyword was added to JavaScript in 2015.
2. Both let and var declare variables but with differing scope. A JavaScript variable's ***scope*** is the context in which the variable can be accessed.
3. A variable declared inside a function has ***local scope***, so only the function that defines the variable has access to the ***local variable***. A variable declared outside a function has ***global scope***, and all functions have access to a ***global variable***.
4. A variable declared inside a function with var has ***function scope***: the variable is accessible anywhere within the function, but not outside. A variable declared inside a function with let has ***block scope***: the variable is accessible only within the enclosing pair of braces.
5. A variable declared using var or let that is not inside a function creates a global variable that is accessible from anywhere in the code.

Table

Description automatically generated with medium confidence

1. var x = 17; declares x with global scope. x is accessible everywhere, so each console.log(x) statement logs x as 17.
2. The var y declaration exists inside the numbers() function. So both console.log(y) statements inside the function log y as 8.5.
3. Similarly, the var z statement is inside the function, so both console.log(z) statements inside the function log z as 34.
4. y and z are not accessible outside the numbers() function. The console.log() statements that exist outside the function throw a ReferenceError when executed.
5. Code that uses let instead of var has similar behavior for the global variable x.
6. The first log statement for y is in y's scope (yellow), and the first log statement for z is in z's scope (green). So, 8.5 and 34 are logged.
7. All remaining calls to log y or z are out of scope and throw a ReferenceError.

**Global variables and the global object**

Before developer code is run, JavaScript implementations create ***the global object***: an object that stores certain global variables, functions, and other properties. When running JavaScript code in a web browser, global variables are usually assigned as properties to the global window object. Therefore, a global variable called test is accessible as window.test.

Developers must be careful when assigning global variables, because a global variable could replace an existing window property. Ex: window.location contains the URL the browser is displaying. Assigning location = "Texas" causes the web browser to attempt to load a web page with the URL "Texas", which likely does not exist.

Three cases exist when assigning to a global variable X:

* X has been declared with var, in which case a property named "X" is added to the global object.
* X has been declared with let, in which case a property named "X" is not added to the global object, but X is still accessible from anywhere in the code.
* X has not been declared with var or let, in which case the variable becomes a property of the global object, even if assigned to inside a function.
* Example with accidental global variable.
* function calculateTax(total) {
* *// Missing "var" so tax becomes a global variable!*
* tax = total \* 0.06;
* return tax;
* }
* var totalTax = calculateTax(10);
* *// tax is accessible because tax is global*
* console.log(tax);
* 0.6

Good practice is to always declare variables used in functions with var or let, so the variables do not become global.