JS

JavaScript Fundamentals

By:Ahmed ElMahdy

Array

What is an Array?

- complex variables that allow us to store more than one value or a group of values under a single variable name.
- can store any valid value, including strings, numbers, objects, functions, and even other arrays.
- Possibility to create more complex data structures such as an array of objects or an array of arrays.
- providing an ordered structure for storing multiple values or a group of values.

Creating an Array

- enclosing a comma-separated list of values in square brackets ([]), as shown in the following syntax:
  var myArray = [*element0, element1, ..., elementN*];
- can also be created using the Array() constructor as shown in the following syntax. (However, for the sake of simplicity previous syntax is recommended).
  var myArray = new Array(*element0, element1, ..., elementN*);

Array

Accessing the Elements of an Array

- can be accessed by their index using the square bracket notation.
- An index is a number that represents an element's position in an array.
- Array indexes start at 0 and go up to the number of elements minus 1. So, array of five elements would have indexes from 0 to 4.

Note: Note: In JavaScript, arrays are really just a special type of objects which has numeric indexes as keys. The typeof operator will return "object" for arrays.

Getting the Length of an Array

- The length property returns the length of an array, which is the total number of elements contained in the array.
- Array length is always greater than the index of any of its element.

JS

Array

Looping Through Array Elements

- You can use for loop to access each element of an array in sequential order.
- ECMAScript 6 has introduced a simpler way to iterate over array element, which is for-of loop. In this loop you don't have to initialize and keep track of the loop counter variable (i).
- can also iterate over the array elements using for-in loop.

Note: The for-in loop should not be used to iterate over an array where the index order is important. The for-in loop is optimized for iterating over object's properties, you should better use a for loop with a numeric index or for-of loop.

Adding New Elements to an Array

- To add a new element at the end of an array, simply use the push() method.
- To add a new element at the beginning of an array use the unshift() method.
- To add multiple elements at once use the push() and unshift() methods.

JS

Array

Removing Elements from an Array

- To remove the last element from an array you can use the pop() method, it returns the value that was popped out.
- To remove the first element from an array using the shift() method. It returns the value that was shifted out.

Tip: The push() and pop() methods runs faster than unshift() and shift(). Because push() and pop() methods simply add and remove elements at the end of an array therefore the elements do not move, whereas unshift() and shift() add and remove elements at the beginning of the array that require re-indexing of whole array.

JS

Object

What is an Object?

- JavaScript is an object-based language
- in JavaScript almost everything is an object or acts like an object.
- A JavaScript object is just a collection of named values. These named values are usually referred to as properties of the object.
- If you remember from the JavaScript arrays chapter, an array is a collection of values, where each value has an index (a numeric key) that starts from zero and increments by one for each value.
- An object is similar to an array, but the difference is that you define the keys yourself, such as name, age, gender, and so on.

In the following sections we'll learn about objects in detail.

Object

Creating Objects

- can be created with curly brackets {} with an optional list of properties.
- A property is a "key: value" pair, where the key (or property name) is always a string.
- value (or property value) can be any data type, like strings, numbers, Booleans or complex data type like arrays, functions, and other objects.
- properties with functions as their values are often called methods to distinguish them from other properties.
- The property names generally do not need to be quoted unless they are reserved words, or if they contain spaces or special characters (anything other than letters, numbers, and the _ and $ characters), or if they start with a number.

Note: Since ECMAScript 5, reserved words can be used as object's property names without quoting. However, you should avoid doing this for better compatibility.

JS

Object

Accessing Object's Properties

- To access or get the value of a property, you can use the dot (.), or square bracket ([]) notation.
- The dot notation is easier to read and write, but it cannot always be used.
- If the name of the property is not valid (i.e. if it contains spaces or special characters), you cannot use the dot notation; you'll have to use bracket notation.
- The square bracket notation offers much more flexibility than dot notation. It also allows you to specify property names as variables instead of just string literals.

Looping Through Object's Properties

Use the for...in loop. This loop is specially optimized for iterating over object's properties.

Setting Object's Properties

Set the new properties or update the existing one using the dot (.) or bracket ([]) notation

Object

Deleting Object's Properties

- The delete operator can be used to completely remove properties from an object.
- It is the only way to actually remove a property from an object.
- Setting the property to undefined or null only changes the value of the property. It does not remove property from the object.

Note: The delete operator only removes an object property or array element. It has no effect on variables or declared functions. However, you should avoid delete operator for deleting an array element, as it doesn't change the array's length, it just leaves a hole in the array.

Calling Object's Methods

Access an object's method the same way as access properties—using the dot notation or using the square bracket notation.

Object

Manipulating by Value vs. Reference

- JavaScript objects are reference types
- Making copies of them means copying the references to that object.
- Whereas primitive values like strings and numbers are assigned or copied as a whole value.

Const object

An object declared as const can be changed but can not reassign.

Cloning and merging, Object.assign

One of the fundamental differences of objects vs primitives is that they are stored and copied “by reference”

Object “this”

- Arrow functions have no “this”
- Arrow functions are special: they don’t have their “own” this.If we reference this from such a function, it’s taken from the outer “normal” function

JS

Symbols

- A “symbol” represents a unique identifier.
- A value of this type can be created using Symbol()

JS

Loops

Different Types of Loops in JavaScript

- used to execute the same block of code again and again, as long as a certain condition is met.
- The idea behind a loop is to automate the repetitive tasks within a program to save the time and effort.

JavaScript now supports five different types of loops:

• **while** — loops through a block of code as long as the condition specified evaluates to true.
• **do…while** — loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.
• **for** — loops through a block of code until the counter reaches a specified number.
• **for…in** — loops through the properties of an object.
• **for…of** — loops over iterable objects such as arrays, strings, etc.

In the following sections, we will discuss each of these loop statements in detail.

JS

Loops

Types of Loops in JavaScript

The while Loop

- This is the simplest looping statement provided by JavaScript.
- It loops through a block of code as long as the specified condition evaluates to true.
- As soon as the condition fails, the loop is stopped.

The generic syntax of the while loop is:

```
while(condition) {
    // Code to be executed
}
```

Note: Make sure that the condition specified in your loop eventually goes false. Otherwise, the loop will never stop iterating which is known as infinite loop. A common mistake is to forget to increment the counter variable (variable i in our case)

Loops

Types of Loops in JavaScript

The do...while Loop

- It is a variant of the while loop.
- Evaluates the condition at the end of each loop iteration.
- With a do-while loop the block of code executed once, and then the condition is evaluated.
- If the condition is true, the statement is repeated as long as the specified condition evaluated to is true.

The generic syntax of the while loop is:

```
do {
    // Code to be executed
}
while(condition);
```

Difference Between while and do...while Loop

- with a while loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.
- With a do-while loop, on the other hand, the loop will always be executed once even if the conditional expression evaluates to false, because unlike the while loop, the condition is evaluated at the end of the loop iteration rather than the beginning.

JS

Loops

The for Loop

- It repeats a block of code as long as a certain condition is met.
- It is typically used to execute a block of code for certain number of times. Its syntax is:

```
for(initialization; condition; increment) {
    // Code to be executed
}
```

- The parameters of the for loop statement have following meanings:
  - ***initialization*** — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.
  - ***condition*** — it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute. If it evaluates to false, the execution of the loop ends.
  - ***increment*** — it updates the loop counter with a new value each time the loop runs.
- The for loop is particularly useful for iterating over an array.

JS

Loops

The for...in Loop

- It is a special type of a loop that iterates over the properties of an object, or the elements of an array. The generic syntax of the for-in loop:

```
for(variable in object) {
    // Code to be executed
}
```

- The loop counter i.e. variable in the for-in loop is a string, not a number.
- It contains the name of current property or the index of the current array element.
- you can loop through the elements of an array

Note: The for-in loop should not be used to iterate over an array where the index order is important. You should better use a for loop with a numeric index.

Loops

The for...of Loop ES6

- ES6 introduces a new for-of loop which allows us to iterate over arrays or other iterable objects (e.g. strings) very easily.
- The code inside the loop is executed for each element of the iterable object.

Note: The for...of loop doesn't work with objects because they are not iterable. If you want to iterate over the properties of an object you can use the for-in loop.

JS

Loops

Skipping parts
Any part of for can be skipped. We can also remove the step part.

Breaking the loop
Normally, a loop exits when its condition becomes falsy. But we can force the exit at any time using the special *break* directive**.**

Continue to the next iteration
It doesn’t stop the whole loop. Instead, it stops the current iteration and forces the loop to start a new one (if the condition allows).

Function

What is Function?

- A function is a group of statements that perform specific tasks and can be kept and maintained separately form main program.

- Functions provide a way to create reusable code packages which are more portable and easier to debug. Here are some advantages of using functions:
  -reduces the repetition of code within a program.
  - makes the code much easier to maintain.
  -makes it easier to eliminate the errors.

Function

Defining and Calling a Function

- The declaration of a function start with the function keyword, followed by the name of the function you want to create, followed by parentheses i.e. () and finally place your function's code between curly brackets {}. Here's the basic syntax for declaring a function:

```
function functionName() {
    // Code to be executed
}
```

- Once a function is defined it can be called (invoked) from anywhere in the document, by typing its name followed by a set of parentheses

Note: A function name must start with a letter or underscore character not with a number, optionally followed by the more letters, numbers, or underscore characters. Function names are case sensitive, just like variable names.

Function

Adding Parameters to Functions

- You can specify parameters when you define your function to accept input values at run time.
- The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.
- Parameters are set on the first line of the function inside the set of parentheses, like this:

```
function functionName(parameter1, parameter2, parameter3) {
    // Code to be executed
}
```

- define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called, otherwise its value becomes undefined.

Function

Default Values for Function Parameters ES6

With ES6, now you can specify default values to the function parameters. This means that if no arguments are provided to function when it is called these default parameters values will be used. This is one of the most awaited features in JavaScript.

Returning Values from a Function

- A function can return a value back to the script that called the function as a result using the return statement. The value may be of any type, including arrays and objects.
- The return statement usually placed as the last line of the function before the closing curly bracket and ends it with a semicolon.
- A function can not return multiple values. However, you can obtain similar results by returning an array of values.

Function

Working with Function Expressions

- The syntax that we've used before to create functions is called function declaration. There is another syntax for creating a function that is called a function expression.
- Once function expression has been stored in a variable, the variable can be used as a function.

Note: There is no need to put a semicolon after the closing curly bracket in a function declaration. But function expressions, on the other hand, should always end with a semicolon.

Tip: In JavaScript functions can be stored in variables, passed into other functions as arguments, passed out of functions as return values, and constructed at run-time.

- The syntax of the *function declaration* and *function expression* looks very similar, but they differ in the way they are evaluated.
- JavaScript parse declaration function before the program executes. Therefore, it doesn't matter if the program invokes the function before it is defined because JavaScript has hoisted the function to the top of the current scope behind the scenes. The function expression is not evaluated until it is assigned to a variable; therefore, it is still undefined when invoked

JS

Function

Understanding the Variable Scope

- you can declare the variables anywhere in JavaScript. But, the location of the declaration determines the extent of a variable's availability within the JavaScript program i.e. where the variable can be used or accessed. This accessibility is known as variable scope.
- By default, variables declared within a function have local scope that means they cannot be viewed or manipulated from outside of that function.
- any variables declared in a program outside of a function has global scope i.e. it will be available to all script, whether that script is inside a function or outside.

JS

Function

Expression vs Declaration

- Function Declaration: a function, declared as a separate statement, in the main code flow.
- Function Expression: a function, created inside an expression or inside another syntax construct. Here, the function is created at the right side of the “assignment expression” =:
- A Function Declaration can be called earlier than it is defined.
- A Function Expression is created when the execution reaches it and is usable only from that moment.

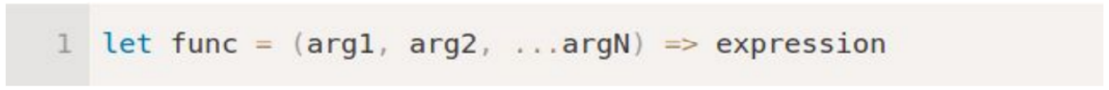
Callback functions

- question: Text of the question
- yes : Function to run if the answer is “Yes”
- no: Function to run if the answer is “No”
- The function should ask the question and, depending on the user’s answer, call yes() or no():

Function

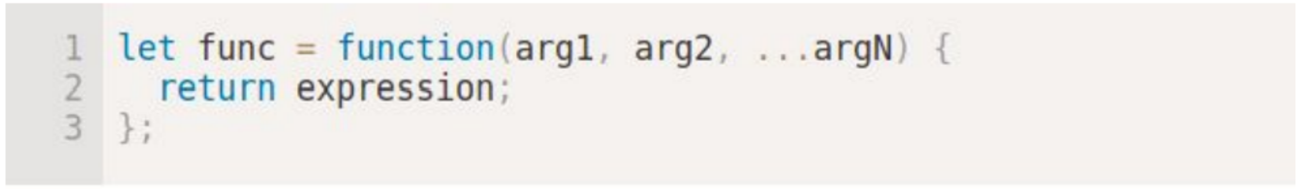
Arrow functions, the basics

- There’s another very simple and concise syntax for creating functions, that’s often better than Function Expressions.
- It’s called “arrow functions”, because it looks like this:



```
let func = (arg1, arg2, ...argN) => expression
```

- This creates a function func that accepts arguments arg1..argN, then evaluates the expression on the right side with their use and returns its result.
- In other words, it’s the shorter version of:



```
let func = function(arg1, arg2, ...argN) {
  return expression;
};
```

- If we have only one argument, then parentheses around parameters can be omitted, making that even shorter.
- If there are no arguments, parentheses will be empty (but they should be present).

JS

Function

Multiline Arrow functions

Sometimes we need something a little bit more complex, like multiple expressions or statements. It is also possible, but we should enclose them in curly braces. Then use a normal return within them.