**Scope**

The three types of variable scopes in JavaScript are global, function, and block.

**Global**

Global scope refers to data stored in variables that are accessible anywhere in your program. Globally scoped variables are declared outside of a function, but are available inside any of the functions in your program.

var cars = ["Audi", "BMW", "Ford"];

function getCars(){

console.log(cars);

}

getCars();

>>["Audi", "BMW", "Ford"];

console.log(cars);

>>["Audi", "BMW", "Ford"];

**Function**

Function (or local) scope refers to data stored in variables that are declared inside of a function and are only available to be accessed inside of the said function.

function getCars(){

var cars = ["Audi", "BMW", "Ford"];

console.log("cars");

}

getCars();

>>["Audi", "BMW", "Ford"];

console.log("cars");

>>Uncaught ReferenceError: cars is not defined

**Block**

Block scope refers to variables which are declared using the keyword let, and are only available inside the block of code inside the function in which they are declared.

function foo(){

var bar = 1;

if(bar < 10){

let baz = "hello!"

console.log(baz);

bar++;

}else{

let baz = "goodbye."

console.log(baz);

}

}

>>hello!

function foo(){

var bar = 10;

if(bar < 10){

let baz = "hello!"

console.log(baz);

bar++;

}else{

let baz = "goodbye."

console.log(baz);

}

}

>>goodbye.

**Primitive types**

In JavaScript, a primitive (primitive value, primitive data type) is data that is not an object and has no methods. The 6 primitive data types are string, number, bigint, Boolean, undefined, and symbol. There is also null.

All primitives are immutable, i.e., they cannot be altered. It is important not to confuse a primitive itself with a variable assigned a primitive value. The variable may be reassigned a new value, but the existing value can not be changed in the ways that objects, arrays, and functions can be altered.

**Factory model**

When creating objects, it is much faster to make a "blueprint" of the object and use that to create many different copies of the object with different parameters. For example, if you want to make a graph with many different coordinate points on it, it is faster to write a function which generates those points with the parameters you put into it, than to write new instructions on how to make a point every time you wish to create one.

const graphPoints = [];

const makeCoordinatePoint = function makeCoordinatePoint(x,y){

let point = {xCoordiate:x, yCoordinate:y};

return point;

}

let pointOne = makeCoordinatePoint(3, 4);

graphPoints.push(pointOne);

console.log(JSON.stringify(graphPoints));

>>[{"xCoordiate":3,"yCoordinate":4}]

let pointTwo = makeCoordinatePoint(7, 24);

graphPoints.push(pointTwo);

console.log(JSON.stringify(graphPoints));

>>[{"xCoordiate":3,"yCoordinate":4},{"xCoordiate":7,"yCoordinate":24}]

**Sources:**

MITxPro BootCamp Course

https://executive-ed.xpro.mit.edu/professional-certificate-coding?utm\_source=Google&utm\_medium=c&utm\_term=%2Bmit%20%2Bcoding&utm\_location=9007574&utm\_campaign=B-365D\_US\_GG\_SE\_PCC\_Brand&utm\_content=MIT-Coding\_\_\_School\_Duration&gclid=Cj0KCQjw3duCBhCAARIsAJeFyPXlm3yGR1CF75aYsJYv3rZzAvneXz0QmWyvSyYu1zLQQfPrdIWiAU4aAqzxEALw\_wcB

[MDN](https://developer.mozilla.org/en-US/)