Differnce between Let and var :

They are very similar when used like this outside a function block.

### Global:

let me = 'go'; // globally scoped

var i = 'able'; // globally scoped

However, global variables defined with let will not be added as properties on the global windowobject like those defined with var.

console.log(window.me); // undefined

console.log(window.i); // 'able'

### Function:

They are identical when used like this in a function block.

function ingWithinEstablishedParameters() {

let terOfRecommendation = 'awesome worker!'; //function block scoped

var sityCheerleading = 'go!'; //function block scoped

}

Block:

function allyIlliterate() {

//tuce is \*not\* visible out here

for( let tuce = 0; tuce < 5; tuce++ ) {

//tuce is only visible in here (and in the for() parentheses)

//and there is a separate tuce variable for each iteration of the loop

}

//tuce is \*not\* visible out here

}

function byE40() {

//nish \*is\* visible out here

for( var nish = 0; nish < 5; nish++ ) {

//nish is visible to the whole function

}

//nish \*is\* visible out here

}

Redeclaration:

Assuming strict mode, var will let you re-declare the same variable in the same scope. On the other hand, let will not:

'use strict';

let me = 'foo';

let me = 'bar'; // SyntaxError: Identifier 'me' has already been declared

'use strict';

var me = 'foo';

var me = 'bar'; // No problem, `me` is replaced.

Arrays

Javascript Array is an object that represents a collection of similar type of element there are 3 ways to construct array in javascritp.

Arrays are collection and are Zero indexed.

Note: Retreving array elements using the array index, will not change the length of the array.

Var myArray = [ ];

for(var i = 0; i<=5; i++){

myArray[i] = I \* 2;

}

for(var i=0; i <= 5; i++ ){

document.write(myArray[i] + “</br>”);

}

alert(myArray.length);

Push method(): Adds new items to the end of the array. This method also changes the length of the array.

Pop method(): Removes the last element of an array. And returns that element. This method changes the length of the array.

Note: Push() and Pop() methods changes the length property of the array.

Unshift method(): push method adds new element to the end of the array. To add new item to the beginning of an array, then use unshif(). Just like pushmethod.

Mutators methos(): The methods that modify the array object are called as mutator method.

Non-mutator methods: contains, indexof, lastindexof.

Mutators methods : Push, Pop, Shift, Unshift, reverse, Sort, splice.

Sort method():

Reverse method(): reverse the order of the elements in an array.

Splice method(): Add or remove elements from an array.

Filter method(): The filter method creates a new array and populates that array with all the elements that meet the condition specified in a callback function.

The filter method calls the callback function one time for each element in the array. If the callback function returns false for all elements of the array, the length of the new array that will be returned Is 0.

Two Dimensional Arrays in Javascript: Javascript does not hve a special syntax for creating multidimensional arrays. Instead we create an array of arrays.

Var myArray1 = new Array(3);

For(I =0; I < 3; I ++){

myArray1[i] = new Array(3);

myArray1[0][1] = “1”; myArray1[1][0] = “4”; myArray[2] [0] = “7”;

myArray1[0][1] = “2”; myArray1[1][1] = “5”; myArray[2] [1] = “8”;

myArray1[0]2] = “3”; myArray1[1][2] = “6”; myArray[2] [2] = “9”;

}

For(var i = 0; i < 3; i ++)

{

For(var j = 0; j < 3; j++)

{

Document.write(myArray1[i][j] + “&emsp;”);

}

Document.write(“</br>”);

}

Functions

A functions is a block of reusable code. A function allows us ot write code once and use it as many times as we want just by calling the function.

Calling the javascript function: Call the javascript function by specifying the name of the function and values for the parameters if any.

What happen when you do not specify values for the function parameters when calling the function: The parameters that are missing values are set to undefined.

What happens when you specify too many parameters values when calling the function: The extra parameters are ignored.

Function Hosting: By default javascript moves all the function declarations to the top of the current scope. This is called function hoisting. This is the reason javascript functions can be called before they are defined.

Defining a function first and the calling it:

function addNumbers(firstNumber, secondNumber){

var result = firstNumber + secondNumber;

return result;

}

var sum = addNumbers(10, 20);

Document.write(sum);

A function call is present before the function definition:

var sum = addNumbers(10, 20);

document.write(sum);

function addNumbers(firstNumber, secondNumber){

Var result = firstNumber + secondNumber;

return result;

}

Defining a javascript function using a function expression: A function expression allows us to define a function using an expression (typically by assigning it to a variable). There are 3 different ways of defining a function using a function expression.

Anonymous function expression: A function have no name that is called a anonymous function. To assign this function to a variable that is called a anonymous function expression.

Local and Global variables in Javascript: In js there are 2 types of variables, these are local and global

Local Variables: local variables are the variables with ina function. Those variables have local scope meaning these are available only inside the function that contains them. Local variables are created when a function starts, and deleted as soon as function completes execution.

Function helloWorld(){

var greeting = ‘Hello’;

greeting = greeting + “Javascript”;

alert(greeting);

}

helloWorld();

document.write(greeting);

Global Variables: Gloabal variables are the variables declared outside a function. Global variables have global scope meaning all scripts and functions on the page can access them. The life time of a global variable starts with it’s declaration and deleted when the page is closed.

Var greeting = ‘Hello’;

function helloWorld(){

greeting = greeting + “Javascript”;

document.write(greeting);

alert(greeting);

}

helloWorld();

document.write(greeting);

if you assing a value to a variable that has not been declared it will automatically become a global

variable, even if it is present inside a function.

Closure: A closure is an inner function that has access to the outer function’s variables in addition to it’s own variables and global variables. The inner function has access not only to the outer function’s variables also to the outer function’s parameters. You create a closure by adding a function inside another function.

function addNumbers(firstNumber, secondNumber){

var return value = “Result is: ”;

function add(){

return returnValue + (firstNumber + secondNumber);

}

return add();

}

var result = addNumbers(10, 20);

document.write(result);

[Every time we click a button on a web page we want to increment the click count by 1. There are several ways we can do this in javascript. One of the way is using global variable]

<script>

Var clickCount = 0;

</script>

<input type=”button” value=”Click Me” onclick=”alert(++clickCount);” />

Every time we click the button the value is incremented as expected. But the problem with this approach is that here clilckCount is a global variable that means any script on the page can access this variable and change this value accidently we want to protect this variable. To protect this we make a local variable with in a function.

function incrementClickCount(){

var clickCount = 0;

Return ++clickCount;

}

<input type=”button” value=”Click Me” onclick=”alert(incrementClickCount());”>

For this approach every time we click a button we calling incrementClickCount function it shows 1 it will not increases. For overcome this we use clouser function.

var incrementClickCount = (function(){

var clickCount = 0;

return function(){

return ++clickCount;

}

})();

<input type=”button” value=”ClickMe” onclick=”alert(incrementClickCount());”>

Recursive function:

Recursive is a programming concept that is applicable to all programming languages including javascript. What is recursive function?

Recursive function is function that calls itself. When writing recursive functions there must be definite break condition. Otherwise we risk creating infinite loops.

// computing the factorial of a number without using recursion

function factorial(n){

If (n == 0 || n == 1){

return 1;

}

var result = n;

while(n > 1){

result = result \* (n - 1)

n = n – 1;

}

return result;

}

Document.write(factorial(5));

Events in Javascript:

What is an event  
An event is a signal from the browser that something has happened. For example,   
1. When a user clicks on an HTML element, click event occurs  
2. When a user moves the mouse over an HTML element, mouseover event occurs  
  
When events occur, we can execute JavaScript code or functions in response to those events. To do this we need to associate JavaScript code or functions to the events. The function that executes in response to an event is called event handler.  
  
In JavaScript, there are several ways to associate an event handler to the event  
1. Using the attributes of an HTML tag  
2. Using DOM object property  
3. Using special methods  
  
In this video we will discuss associating event handler methods to events using the attributes of HTML tags.   
  
In the following example, the code to execute in response to onmouseover & onmouseout events is set directly in the HTML markup. The keyword "this" references the current element. In this example "this" references the button control.

What is DOM?

Dom stands for Document Object Model. When a browser loads a web page, the browser creates a Document object model that page.

JavaScript can be used to access and modify these DOM objects and their properties. For example, you can add, modify and remove HTML elements and their attributes. Along the same lines, you can use DOM object properties to assign event handlers to events.

EventBubbling in Javascript

Event bubbling process starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy.

Javascript OOP concept

JavaScript is object oriented programming language. The following are the 4 pillars of any object oriented programming language. We will discuss examples of these in a later video session.  
1. Inheritance  
2. Encapsulation  
3. Abstraction  
4. Polymorphism  
  
In this video let's focus on creating objects in JavaScript. Objects in JavaScript can be broadly classified into 2 categories.  
1. Standard built-in objects  
2. Custom objects  
  
Standard built-in objects : So far in this video series, we have already seen many of the JavaScript standard built-in objects. Examples include string, array, RegExp, Date etc. In the example below we are creating Date object using the Date constructor function and then using it's getFullYear() method to get the year.  
  
var currentDate = new Date();  
document.write(currentDate.getFullYear());  
  
Custom objects : In C#, to create a custom object, we create a Custom class and then create an instance of a class. In JavaScript we don't have classes. Instead we use functions. In JavaScript there are two ways to create a custom object.  
1. Constructor function   
2. Literal notation

What is the difference between creating an object using constructor function and literal notation.   
1. In the constructor function the properties and their values separated using an equal-to sign(=) whereas in the literal version, they are separated using a colon (:)  
2. In constructor function at the end of each property you can have a semi-colons (;) whereas in the literal version properties must be separated with a comma (,)  
3. With literal notation you have already created an object, so to access firstName value you simply use employee.firstName. With the constructor function you have to first create an instance and then use the created instance and the property name separated by DOT as shown below.  
  
var employee = new Employee("Pragim", "Tech");  
employee.firstName.  
Objects created using object literals are singletons. This means when a change is made to the object, it affects that object across the entire script.

Object defined with a function constructor lets you have multiple instances of that object. This means change made to one instance, will not affect other instances.  
  
So, when to use one over the other?  
If you need multiple instances of the object use constructor function where as if you need just one instance of the object then use literal notation.

Global Namespace pollution in Javascript:

When working with javascript in large projects during your code review you might hear your senior developer says this javascript code causes the global scope to be polluted.

Polluting global namespace causes name collision. This is especially true in large projects where you may be using several JavaScript libraries (both internally developed as well as third party libraries). That's why it is very important not to add everything to the global namespace. If someone else use the same variable or function names it can lead to name collision.

Properties in Javascript:

In an OOP language , classes can have properties. For example a class in c# can have the following properties.

Read/Write properties

Read only properties

Write only properties

Since JavaScript is also an object oriented programming language, objects in JavaScript can also have properties.   
  
Why do we need properties when we have public fields  
Encapsulation is one of the pillars of object oriented programming language. Properties provide encapsulation. If you use public fields you cannot control on what is assigned and returned from that public field.  
  
In the example below we have an employee object with age public field. There is nothing stopping us from setting the age value of the employee object to 1000. Using properties you can control on what values can be assigned. You can also use properties to create just read-only or write-only properties.

Inheritance in Javascript

In Object oriented programming languages like C# and Java to implement inheritance, a class inherits from another class. In JavaScript, we don't have the concept of classes, so inheritance in JavaScript is prototype-based. This means to implement inheritance in JavaScript, an object inherits from another object

In simple terms, inheritance is the concept of one thing gaining the properties or behaviours of something else. To say A **inherits** from B, is saying that A **is a type of** B. A Bird **inherits** from Animal because a Bird **is a type of** Animal - it can do the same things, but a little more (or differently)!

In JavaScript, this relationship is a little complicated to specify, but bear with the syntax. You must use a special object called prototype which assigns properties to a **type** such as Animal. Only functions have a prototype, which is why you must create a function first:

Abstract in Javascript  
Object oriented programming languages like C# and Java, support abstract classes. Abstract classes are incomplete. So, trying to create an instance of an abstract class raises a compiler error. Abstract classes can only be used as base classes.

Conditional statements : JavaScript code is executed in a linear fashion from the first line to the last line. If for some reason you want to interrupt this flow and execute certain statements, only, if certain condition is met, then we use conditional statements.

JavaScript has the following conditional statements if, if else, if else if, else, switch, ternary operator - shortcut for an if...else statement

Switch statements: To improve the readability of a program multiple if else if statements can be replaced with a switch statements.

**String:** A string is an any text inside quotes. You can use either single or double quotes. There are two options to concatenate strings in javascript.

Using + operator, or concatenate method

Note: Refference site : https://www.codeproject.com/Articles/618484/Latest-jQuery-interview-questions-and-answers

JQUERY

Difference between $(document).ready and $(window).load:

$(document).ready is a jquery even. It fires as soon as the DOM is loaded and ready to be manipulated by script. This is earliest point in the page load process where the script can safely access element’s in the page’s html dom. This event is fired before all the images etc

$(window).load event fires when then DOM and all the content on the page (images, css etc) is fully loaded. Since the window load event waits for images, css etc to be fully loaded, this event fires after ready event.

In most cases, the script can be run as soon as the DOM hierarchy has been fully constructed. So ready() is usually the best place to write your JavaScript code.

However, in your application there could be scenarios where you should be using $(window).load over $(document).ready. For example, let's say we want to display the actual image dimensions (Height and Width). To get the actual image dimensions, we will have to wait until the image is fully loded, so the jQuery code to get the height and width should be in $(window).load event.

What are Jquery selectors:

One of the most important concept in jquery is selectors. jQuery selectors allows you to select and manipulate HTML elements.

Different selectors in jquery

1Element ID  
2. Element Tag Name  
3. Element Class Name  
4. Element attribute  
5. Element Attribute values and many more

Id selector in jquery  
To find an HTML element by ID, use the jQuery [#id](https://www.youtube.com/results?q=%23id) selector

jQuery input vs input

$(:input) selects all input, textarea, select and button elements whereas $(input) just selects elements with an input tag.

The :checked selector selects all checked checkboxes or radio buttons.