# Assignment Day 1 | 14th July 2020

Ankit Ramprakash Sharma

## Question 1:

💡 Explore and explain the various methods in console function Explain them Ex. console.log() console.warn(). etc...

A Web console is a tool which is mainly used to log information associated with a web page like: network requests, javascript, security errors, warnings, CSS etc. It enables us to interact with a web page by executing javascript expression in the contents of the page. The Console object provides access to the browser's debugging console. We can open a console in web browser by using F12 key in windows.

**Methods:- 1. console.log()** For general output of logging information. You may use string substitution and additional arguments with this method.

**2. console.warn()** Outputs a warning message. You may use string substitution and additional arguments with this method.

**3. console.error()** Outputs an error message. You may use string substitution and additional arguments with this method.

**4. console.table()** Displays tabular data as a table.

**5. console.time()** Starts a timer with a name specified as an input parameter. Up to 10,000 simultaneous timers can run on a given page.

**6. console.timeEnd()** Stops the specified timer and logs the elapsed time in seconds since it started.

**7. console.clear()** Clear the console.

**8. console.count()** Log the number of times this line has been called with the given label.

**9. console.group()** Creates a new inline group, indenting all following output by another level. To move back out a level, call groupEnd().

**10. console.groupCollapsed()** Creates a new inline group, indenting all following output by another level. However, unlike group() this starts with the inline group collapsed requiring the use of a disclosure button to expand it. To move back out a level, call groupEnd().

**Execution of these methods are available in the HTML file.**

A Web console is a tool which is mainly used to log information associated with a web page like: network requests, javascript, security errors, warnings, CSS etc. It enables us to interact with a web page by executing javascript expression in the contents of the page. The Console object provides access to the browser's debugging console. We can open a console in web browser by using F12 key in windows.

## Question 2:

👉 Write the difference between var, let and const with code examples.

In Javascript one can define variables using the keywords **var, let** or **const**. Eg: var a=10; let b=20; const PI=3.14;

**var:** The scope of a variable defined with the keyword "var" is limited to the "function" within which it is defined. If it is defined outside any function, the scope of the variable is global. var is "function scoped".

**let:** The scope of a variable defined with the keyword "let" or "const" is limited to the "block" defined by curly braces i.e. {} . "let" and "const" are"block scoped".

**const:** The scope of a variable defined with the keyword "const" is limited to the block defined by curly braces. However if a variable is defined with keyword const, it cannot be reassigned. "const" cannot be re-assigned to a new value. However it CAN be mutated.

**Function scoped vs. Block scoped** Let us understand this by some code examples,

**Block Scope** In Javascript you can define a code block using curly braces i.e {}. Consider the following code that has 2 code blocks each delimited by {}.

```
{
  var a=10;
  console.log(a);
} //block 1
{
  a++;
  console.log(a);
} //block 2
/* Since we are using "var a=10", scope of "a" is limited to the function
within which it is defined. In this case it is within the global function scope */
```

In the above example, since we are using the keyword var to define the variable a, the scope of a is limited to the function within which it is defined. Since a is not defined within any function, the scope of the variable a is global, which means that a is recognized within block 2. In effect if a variable is defined with keyword var, Javascript does not recognize the {} as the scope delimiter. Instead the variable must be enclosed within a “function” to limit it’s scope to that function. Let us re-write the code above using the keyword let. The let keyword was introduced as part of ES6 syntax, as an alternative to var to define variables in Javascript.

```
{
 let a=10;
 console.log(a);
} //block 1
{
  a++;
  console.log(a);
} //block 2
/* Since we are using "let a=10", scope of "a" is limited to block 1 and "a" is not
recognized in block 2 */
```

Note that now when you run the code above you will get an error, variable a not recognized in block2. This is because we have defined the variable a using the keyword let, which limits the scope of variable a to the code block within which it was defined.

**Function Scope** In Javascript you limit the scope of a variable by defining it within a function. This is known as function scope. The keyword var is function scoped i.e. it does not recognize curly brackets i.e. {}, as delimiters. Instead it recognizes the function body as the delimiter. If we want to define a variable using var, and prevent it from being defined in the global namespace you can re-write it by enclosing the code blocks within functions.

```
function block1() {
var a=10;
 console.log(a);
} //function scope of block 1
function block2() {
  a++;
  console.log(a);
} //function scope of block 2
/* Since we have enclosed block1 and block2, within separate functions,
the scope of "var a=10", is limited to block 1 and "a" is not recognized in block 2 */
```

The above code is in effect the same as if we were using let a=10 instead of var a=10. The scope of the variable a is limited to the function within which it is defined, and a is no longer in the global namespace.

**Why would you chose “let” over “var”?** While programming in Javascript it is a good practice not to define variables as global variables. This is because it is possible to inadvertently modify the global variable from anywhere within the Javascript code. To prevent this one needs to ensure that the scope of the variables are limited to the code block within which they need to be executed. In the past before keyword let was introduced as part of ES6, to circumvent the issue of variable scoping using var, programmers used the IIFE pattern to prevent the pollution of the global name space. However since the introduction of let, the IIFE pattern is no longer required, and the scope of the variable defined using let is limited to the code block within which it is defined.

**The “const” keyword** If a variable is defined using the const keyword, its scope is limited to the block scope. In addition the variable cannot be reassigned to a different value.

```
{
 const PI=3.14;
 console.log(PI);
} //block 1
{
  console.log(PI);
} //block 2
/* Since we are using "const PI=3.14", scope of "PI" is limited to block 1 and "PI" is
not recognized in block 2 */
```

Note that it is important to understand that const does NOT mean that the value is fixed and immutable. This is a common misunderstanding amongst many Javascript developers, and they incorrectly mentioned that a value defined by the const keyword is immutable (i.e. it cannot be changed). In the following example we can show that the value of the variable defined within the const keyword is mutable, i.e. it can be changed.

```
{
 const a = [1,2,3];
 const b = {name: "hello"};
 a.push(4,5);     //mutating the value of constant "a"
 b.name="World";  //mutating the value of constant "b"

 console.log(a); //this will show [1,2,3,4,5]
 console.log(b); //this will show {name: "World"}

}
/* This code will run without any errors, and shows that we
CAN mutate the values that are defined by "const" */
```

However note that these variables defined by const cannot be re-assigned.

```
{
 const name = "Mike";
 const PI = 3.14;
 const a = [1,2,3];
 const b = {name: "hello"};

 name="Joe";
/*this will throw an error, since we are attempting to re-assign "name" to a
 different value.*/
 PI = PI + 1;
/*this will throw an error, since we are attempting to re-assign PI to a
different value.*/
 a = [1,2,3,4,5];
/*this will throw an error, since we are attempting to re-assign "a" to a
different value.*/
 b = {name: "hello"};
/*this will throw an error, since we are attempting to re-assign "b" to a
different value.*/
}
```

## Question 3:

👉 Write a brief intro on available data types in Javascript.

**Data Types in JavaScript:** Data types basically specify what kind of data can be stored and manipulated within a program.

There are six basic data types in JavaScript which can be divided into three main categories: primitive (or primary), composite (or reference), and special data types. String, Number, and Boolean are primitive data types. Object, Array, and Function (which are all types of objects) are composite data types. Whereas Undefined and Null are special data types.

**Primitive data** types can hold only one value at a time, whereas composite data types can hold collections of values and more complex entities. Let's discuss each one of them in detail.

**1. The String Data Type** The string data type is used to represent textual data (i.e. sequences of characters). Strings are created using single or double quotes surrounding one or more characters, as shown below:

```
var a = 'Hi there!';  // using single quotes
var b = "Hi there!";  // using double quotes
/*You can include quotes inside the string as long as they don't match the
enclosing quotes.
*/
var a = "Let's have a cup of coffee."; // single quote inside double quotes
var b = 'He said "Hello" and left.';  // double quotes inside single quotes
var c = 'We\'ll never give up.';     // escaping single quote with backslash
```

**2. The Number Data Type** The number data type is used to represent positive or negative numbers with or without decimal place, or numbers written using exponential notation e.g. 1.5e-4 (equivalent to 1.5×10-4).

```
var a = 25;         // integer
var b = 80.5;       // floating-point number
var c = 4.25e+6;    // exponential notation, same as 4.25e6 or 4250000
var d = 4.25e-6;    // exponential notation, same as 0.00000425
```

The Number data type also includes some special values which are: Infinity, -Infinity and NaN. Infinity represents the mathematical Infinity ∞, which is greater than any number. Infinity is the result of dividing a nonzero number by 0, as demonstrated below:

```
alert(16 / 0);  // Output: Infinity
alert(-16 / 0); // Output: -Infinity
alert(16 / -0); // Output: -Infini
```

While NaN represents a special Not-a-Number value. It is a result of an invalid or an undefined mathematical operation, like taking the square root of -1 or dividing 0 by 0, etc.

**3. The Boolean Data Type** The Boolean data type can hold only two values: true or false. It is typically used to store values like yes (true) or no (false), on (true) or off (false), etc. as demonstrated below:

```
var isReading = true;   // yes, I'm reading
var isSleeping = false; // no, I'm not sleeping
```

Boolean values also come as a result of comparisons in a program. The following example compares two variables and shows the result in an alert dialog box:

```
var a = 2, b = 5, c = 10;
alert(b > a) // Output: true
alert(b > c) // Output: false
```

The Undefined Data Type The undefined data type can only have one value-the special value undefined. If a variable has been declared, but has not been assigned a value, has the value undefined.

```
var a;
var b = "Hello World!"

alert(a) // Output: undefined
alert(b) // Output: Hello World!
```

**4. The Null Data Type** This is another special data type that can have only one value-the null value. A null value means that there is no value. It is not equivalent to an empty string ("") or 0, it is simply nothing. A variable can be explicitly emptied of its current contents by assigning it the null value.

```
var a = null;
alert(a); // Output: null

var b = "Hello World!"
alert(b); // Output: Hello World!

b = null;
alert(b) // Output: null
```

**5. The Object Data Type** The object is a complex data type that allows you to store collections of data. An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, like strings, numbers, booleans, or complex data types like arrays, function and other objects. You'll learn more about objects in upcoming chapters.

The following example will show you the simplest way to create an object in JavaScript.

```
var emptyObject = {};
var person = {"name": "Clark", "surname": "Kent", "age": "36"};

// For better reading
var car = {
    "modal": "BMW X3",
    "color": "white",
    "doors": 5
}
```

You can omit the quotes around property name if the name is a valid JavaScript name. That means quotes are required around "first-name" but are optional around firstname. So the car object in the above example can also be written as:

```
var car = {
    modal: "BMW X3",
    color: "white",
    doors: 5
}
```

**6. The Array Data Type** An array is a type of object used for storing multiple values in single variable. Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type- numbers, strings, booleans, functions, objects, and even other arrays. The array index starts from 0, so that the first array element is arr[0] not arr[1].

The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:

```
var colors = ["Red", "Yellow", "Green", "Orange"];
var cities = ["London", "Paris", "New York"];

alert(colors[0]);   // Output: Red
alert(cities[2]);   // Output: New York
```

**7. The Function Data Type** The function is callable object that executes a block of code. Since functions are objects, so it is possible to assign them to variables, as shown in the example below:

```
var greeting = function(){
    return "Hello World!";
}
// Check the type of greeting variable
alert(typeof greeting) // Output: function
alert(greeting());     // Output: Hello World!
```

In fact, functions can be used at any place any other value can be used. Functions can be stored in variables, objects, and arrays. Functions can be passed as arguments to other functions, and functions can be returned from functions. Consider the following function:

```
function createGreeting(name){
    return "Hello, " + name;
}
function displayGreeting(greetingFunction, userName){
    return greetingFunction(userName);
}

var result = displayGreeting(createGreeting, "Peter");
alert(result); // Output: Hello, Peter
```