**14th July 2020 Assignment Day 1/Batch 1**

**Question 1: Explore and explain the various methods in console function. Explain them.**

Various methods in console function are:

|  |  |
| --- | --- |
| console.assert() | Log a message and stack trace to console if the first argument is false |
| console.clear() | To clear the console |
| console.count() | Log of number of times a line with the given label has been called. |
| console.countReset() | Resets the value of the counter with the given label. |
| console.debug() | Outputs a message to the console with the log level "debug". |
| console.dir() | It displays an interactive listing of the properties of a specified JavaScript object and this listing lets us to use disclosure triangles to examine the contents of child objects. |
| console.dirxml() | It displays an XML/HTML Element representation of the specified object if possible or the JavaScript Object view if it is not possible. |
| console.error() | It outputs an error message, we can use string substitution and additional arguments with this method. |
| console.exception() | It is an alias for error(). |
| console.group() | It creates a new inline group, indenting all following output by another level. Call groupEnd() to move back out a level. |
| console.groupCollapsed() | To create a new inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_groups_in_the_console), 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. Call groupEnd() to move back out a level. |
| console.groupEnd() | To exit the current inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_groups_in_the_console). |
| console.info() | Informative logging of information. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method. |
| console.log() | For general output of logging information. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method. |
| console.profile() | Starts the browser's built-in profiler (for example, the [Firefox performance tool](https://developer.mozilla.org/en-US/docs/Tools/Performance)). You can specify an optional name for the profile. |
| console.profileEnd() | Stops the profiler. You can see the resulting profile in the browser's performance tool (for example, the [Firefox performance tool](https://developer.mozilla.org/en-US/docs/Tools/Performance)). |
| console.table() | Displays tabular data as a table. |
| console.time() | Starts a [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#Timers) with a name specified as an input parameter. Up to 10,000 simultaneous timers can run on a given page. |
| console.timeEnd() | Stops the specified [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#Timers) and logs the elapsed time in seconds since it started. |
| console.timeLog() | Logs the value of the specified [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#Timers) to the console. |
| console.timeStamp() | Adds a marker to the browser's [Timeline](https://developer.chrome.com/devtools/docs/timeline) or [Waterfall](https://developer.mozilla.org/en-US/docs/Tools/Performance/Waterfall) tool. |
| console.trace() | Outputs a [stack trace](https://developer.mozilla.org/en-US/docs/Web/API/console#Stack_traces). |
| console.warn() | Outputs a warning message. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method. |

**Question 2: Write the difference between var, let and const with code examples.**

var , let and const all are for declaring the variables. Difference among them as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | var | Let | const |
| scope | Globally scoped or function/locally scoped.  Globally scoped: A variable is declared outside a function that is available for the whole window.  Function scoped: It is declared within a function that means available and can be accessed only within that function. | Block scoped:  A block is a chunk of code bounded by {}. A block lives in curly braces. Anything within curly braces is a block.  So a variable declared in a block with let is only available for use within that block. | Variables declared with the const maintain constant values. const declarations share some similarities with let declarations. const declarations are block scoped Like let declarations, const declarations can only be accessed within the block they were declared. |
| Example coding: | **Globally scoped & function scoped:**  var greeter = "hey hi"; // globally scoped  function newFunction()  { var hello = "hello"; // function scoped  }  **Showing error in coding:**  var tester = "hey hi";  function newFunction()  { var hello = "hello"; }  console.log(hello); // error: hello is not defined(not being available outside the function) | let greeting = "say Hi";  let times = 4;  if (times > 3)  { let hello = "say Hello instead";  console.log(hello);// "say Hello instead  }  console.log(hello) // hello is not defined  We see that using hello outside its block (the curly braces where it was defined) returns an error. This is because let variables are block scoped. | const greeting = "say Hi"; |
| Re-declared and updated | **Can be updated or re-declared.**  We can do this within the same scope and won’t get an error.  Example:  var greeter = "hey hi";  var greeter = "say Hello instead";  and this also, var greeter = "hey hi";  greeter = "say Hello instead"; | **let can be updated but not re-declared.**  Just like var,  a variable declared with let can be updated within its scope. Unlike var, a let variable cannot be re-declared within its scope. So while this will work:  let greeting = "say Hi";  greeting = "say Hello instead";  this will return an error:  let greeting = "say Hi";  let greeting = "say Hello instead"; // error: Identifier 'greeting' has already been declared  However, if the same variable is defined in different scopes, there will be no error:  let greeting = "say Hi";  if (true) {  let greeting = "say Hello instead";  console.log(greeting); // "say Hello instead"  }  console.log(greeting); // "say Hi"  Why is there no error? This is because both instances are treated as different variables since they have different scopes.  This fact makes let a better choice than var. When using let, you don't have to bother if you have used a name for a variable before as a variable exists only within its scope.  Also, since a variable cannot be declared more than once within a scope, then the problem discussed earlier that occurs with var does not happen. | **Const cannot be updated or re-declared.**  This means that the value of a variable declared with const remains the same within its scope. It cannot be updated or re-declared. So if we declare a variable with const, we can neither do this:  const greeting = "say Hi";  greeting = "say Hello instead";// error: Assignment to constant variable. nor this:  const greeting = "say Hi";  const greeting = "say Hello instead";// error: Identifier 'greeting' has already been declared  Every const declaration, therefore, must be initialized at the time of declaration.  This behavior is somehow different when it comes to objects declared with const. While a const object cannot be updated, the properties of this objects can be updated. Therefore, if we declare a const object as this:  const greeting = {message: "say Hi",times: 4}  while we cannot do this:  const greeting = {words: "Hello", number: "five"} // error: Assignment to constant variable.  we can do this: greeting.message = "say Hello instead";  This will update the value of greeting.message without returning errors. |
| Hoisting | Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. This means that if we do this:  console.log (greeter);  var greeter = "say hello"  it is interpreted as this:  var greeter;  console.log(greeter); // greeter is undefined  greeter = "say hello"  So var variables are hoisted to the top of their scope and initialized with a value of undefined. | Just like  var, let declarations are hoisted to the top. Unlike var which is initialized as undefined, the let keyword is not initialized. So if you try to use a let variable before declaration, you'll get a Reference Error. | Just like let, const declarations are hoisted to the top but are not initialized. |

**Question 3: Write a brief note 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.

**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:

**Example**

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

**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 1.5x10-4).

**Example**

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 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:

**Example**

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:

**Example**

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.

**Example**

var a;

var b = "Hello World!"

alert(a) // Output: undefined

alert(b) // Output: Hello World!

**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.

**Example**

var a = null;

alert(a); // Output: null

var b = "Hello World!"

alert(b); // Output: Hello World!

b = null;

alert(b) // Output: null

var emptyObject = {};

var person = {"name": "Clark", "surname": "Kent", "age": "36"}; // For better reading

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

**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:

**Example**

var colors = ["Red", "Yellow", "Green", "Orange"];

var cities = ["London", "Paris", "New York"];

alert(colors[0]); // Output: Red

alert(cities[2]); // Output: New York

**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:

**Example**

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

alert(typeof greeting) // Output: function

alert(greeting()); // Output: Hello World!