LetsUpgrade JavaScript Essentials
Assignment-Day1

1. **Explain various methods of console function.**

   In JavaScript, the console is an object which provides access to the browser debugging console. The console object provides us with several different methods. Some of them are:

   log ()

   -Mainly used to log(print) the output to the console. We can put any type inside the log (), be it a string, array, object, boolean etc.

   E.g.:
   ```
   console.log('abc');
   console.log (1);
   console.log(true);
   console.log(null);
   console.log(undefined);
   console.log ([1, 2, 3, 4]);
   console.log ({a:1, b:2, c:3});
   ```

   error ()

   - Used to log error message to the console. Useful in testing of code. By default, the error message will be highlighted with red color.

   E.g.: console. error ('An error occurred');

   warn ()

   - Used to log warning message to the console. By default the warning message will be highlighted with yellow color.

   E.g.: console. warn ('This is a warning');

   clear ()

   - Used to clear the console. The console will be cleared, in case of Chrome a simple overlayer text will be printed like: 'Console was cleared' while in Firefox no message is returned.

E.g.: console. clear ();

time () and timeEnd ()

- The console object has time () and timeEnd () methods that help with analyzing performance of pieces of your code. You first call console.time () by providing a string argument, then the code that you want to test, then call console.timeEnd () with the same string argument. You'll then see the time it took to run the code in your browser console.

E.g.:

```
console.time('abc');
let fun = function () {
    console.log ('fun is running');
}
let fun2 = function () {
    console.log ('fun2 is running...');
}
fun (); // calling fun ();
fun2(); // calling fun2();
console.timeEnd('abc');
```

OUTPUT:

```
fun is running
fun2 is running...
abc: 1ms
```

table ()

- This method allows us to generate a table inside a console. The input must be an array or an object which will be shown as a table.

E.g.: console. table ({'a':1, 'b':2});

count ()

- This method is used to count the number that the function hit by this counting method.

E.g.: for (let i=0; i<5; i++) {

```
    console. count(i);
```

}

group () and groupEnd()

- group () and groupEnd() methods of the console object allows us to group contents in a separate block, which will be indented. Just like the time () and the timeEnd () they also accept label, again of same value.

E.g.:
```
console.group('simple');
    console. Warn('warning!');
    console.error('error here');
    console.log ('logged');
    console.groupEnd('simple');
    console.log ('new section');
```

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

Scope wise difference:

- var keyword is function scoped.
- let and const keywords are block scoped.

E.g.:
```
if (true) {
var foo = 'peekaboo!';
let bar = 'i see u';
const baz = 'baby blue!';
console.log(foo); // 'peekaboo!';
console.log(bar); // 'i see u';
console.log(baz); // 'baby blue!';
}
console.log(foo); // 'peekaboo!';
console.log(bar); // ReferenceError: bar is not defined
console.log(baz); // ReferenceError: baz is not defined
```

The visibility of foo isn't limited by the if-statement block as it is function scoped. However, both bar and baz are limited in visibility to the block of code.

This concept of scope is the most prominent distinction between the old-fashioned var and modern let/const.

Redefinable and Re-declarable:

- var is re-declarable and redefinable.
- let is not re-declarable but redefinable.
- const is neither re-declarable nor redefinable.

E.g.:

```
function myFn() {

var foo = 1;

foo = 30; //var is redefinable

var foo = 101; //var is re-declarable

console.log(foo); //101

let bar = 1;

bar= 30; //let is redefinable

 let bar = 101; // Uncaught SyntaxError: Identifier 'bar' has already been declared

// let is not re-declarable.

bar= 101;

console.log(foo); // 101

const myBoolean = true;

myBoolean=false;// Uncaught TypeError: Assignment to constant variable.

//const is not redefinable and not redefinable

}

myFn();
```

3. **Write a brief intro on available datatypes of JavaScript.**

Primitive Data:
A primitive data value is a single simple data value with no additional properties and methods.

The typeof operator can return one of these primitive types:

- string
- Number
- boolean

- undefined

Example:

```
typeof "John"            // Returns "string"
typeof 3.14              // Returns "number"
typeof true              // Returns "boolean"
typeof false             // Returns "boolean"
typeof x                 // Returns "undefined" (if x has no value)
```

Complex Data

The typeof operator can return one of two complex types:

- function
- object

The typeof operator returns "object" for objects, arrays, and null.

The typeof operator does not return "object" for functions.

Example

```
typeof {name: 'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object"
typeof null                  // Returns "object"
typeof function myFunc() {} // Returns "function"
```