JavaScript ES6

Lesson: 13
New syntax-based features
& New types

Capgemini

Lesson Objectives

At the end of this module you will be able to:

- Creating the block scoped variables using the let keyword
- Creating constant variables using the const keyword
- Use spread operator and the rest parameter
- Extract the data from iterables and objects using the destructuring assignment
- Use arrow functions
- Use new syntactic features, introduced by ES6.

The let keyword

let keyword is used to declare a block scoped variable, optionally initializing it to a value

Variables that are declared using the ***let*** keyword are called as block scoped variables.

Block scoped variables are accessible only inside the block in which it is defined.

let keyword doesn't allow to declare the variable again in the same scope, it will throw error.

No ***hoisting*** will takes place when let keyword is used; i.e. let keyword ensures variable declaration takes place before it is used.

Demo

let-keyword

The const keyword

const keyword is used to declare the read-only variables, i.e. the variables whose value cannot be reassigned.

Constant variables are block-scoped variables, i.e. they follow the same scoping rules as the variables that are declared using the let keyword.

JavaScript object can be assigned to a constant variable, when assigning an object to a constant variable, the reference of the object becomes constant to that variable and not to the object itself. Therefore, the object is mutable.

Demo

const-keyword

The arrow functions

ES6 provides a new way to create functions using the => operator.

Functions created using => operator is called as arrow functions. It can be also called as anonymous functions.

The arrow functions are the instances of the Function constructor.

If an arrow function contains just one statement, no need to wrap the code in brackets {} and the statement in the body is automatically returned.

this keyword used inside the arrow function will return the context of the code in which it is running.

The arrow functions cannot be used as object constructors i.e. the new operator cannot be applied on them.

Demo

arrow-functions

Default parameter values

In JavaScript there is no defined way to assign the default values to the function parameters that are not passed.

Programmers need to check the parameters with undefined value and assign the default values.

ES6 provides a new syntax that can be used to do this in an easier way.

Default values / expression can be assigned with the parameter, which gets overridden if the value is passed unless undefined is passed as value.

Default parameters cannot be used before the declaration

For Dynamic function, the default parameter and body of the function(as last parameter) can be passed as string.

Demo

default-parameter-values

The rest parameter

The rest parameter is represented by the "…" token.

The last parameter of a function prefixed with "…" is called as a rest parameter.

The rest parameter is an array type, which contains the rest of the parameters of a function when number of arguments exceeds the number of named parameters.

If the arguments exactly match with the named parameters, rest parameter returns empty array []

Demo

rest-parameter

The spread operator

The spread operator is also represented by the "…" token.

A spread operator can be placed wherever multiple function arguments or multiple elements (for array literals) are expected in code.

A spread operator splits an iterable object into the individual values

- An iterable is an object that contains a group of values, and implements ES6 iterable protocol to iterate through its values. An array is an example of built in an iterable object

Demo

spread-operator

The for…of loop

The for…of loop is used to iterate over the values of an iterable object.

```
(function(){
var departments = ["Training","HR","BPO"];
    for(var department of departments){
            console.log(department);
                }

        var department = "Training";
 for(var charInDepartment of department){
            console.log(charInDepartment);
                }
        })();
```

Demo

for…of

Template literals

Template literals are string literals allowing embedded expressions.

Template literals are enclosed by the back-tick (` `) (grave accent) character instead of double or single quotes.

It is always processed and converted to a normal JavaScript string on runtime so it can be used in the place of normal strings.

The expressions are placed in placeholders indicated by dollar sign and curly brackets, i.e. ${expressions}

If custom function is used to process the string parts then the template string is called as a tagged template string and the custom function is called as tag function.

Demo

template-literals

Destructuring assignment

Destructuring assignment is an expression that allows to assign the values or properties of an iterable or object, to the variables, using a syntax that looks similar to the array or object construction literals respectively.

There are two kinds of destructuring assignment expressions array and object destructuring assignment.

- An array destructuring assignment is used to extract the values of an iterable object and assign them to the variables. It's named as the array destructuring assignment because the expression is similar to an array construction literal.
- An object destructuring assignment is used to the extract property values of an object and assign them to the variables.

Demo

destructuring

The ES6 symbols

symbols are the new primitive type like the Number, String, and Boolean introduced in ES

A symbol is a unique identifier where the unique identifier can never be accessed.

Symbol() function creates and returns a unique symbol every time it is called.

The Symbol() function takes an optional string parameter that represents the description of the symbol.

A description of a symbol can be used for debugging, but not to access the symbol itself. i.e. Two symbols with the same description are not equal at all.

The primary reason for introducing symbols in ES6 was so that it can be used as a key for object property, and prevent the accidental collision of the property keys.

The ES6 symbols

The Symbol object maintains a registry of the key/value pairs, where the key is the symbol description, and the value is the symbol.

When symbol is created using Symbol.for() method, it gets added to the registry and the method returns the symbol. If a symbol is created with a description that already exists, then the existing symbol will be retrieved.

Symbol.for() method makes the symbol available globally.

ES6 introduced Object.getOwnPropertySymbols() to retrieve an array of symbol properties of an object.

In addition to the custom symbols, ES6 comes up with a built-in set of symbols called as well-known symbols. It is used for meta programming (looking deeper into objects, functions and even how JavaScript engine operates).

Demo

symbol

well-known symbols

Here is a list of properties, referencing some important built-in symbols.

- Symbol.iterator
- Symbol.match
- Symbol.search
- Symbol.replace
- Symbol.split
- Symbol.hasInstance
- Symbol.species
- Symbol.unscopables
- Symbol.isContcatSpreadable
- Symbol.toPrimitive
- Symbol.toStringTag

Summary

Variables that are declared using the let keyword are called as block scoped variables.

for…of loop is used to iterate over the values of an iterable object

spread operator splits an iterable object into the individual values

Templates are processed and converted to a normal JavaScript string on runtime.

A symbol is a unique and immutable data type and may be used as an identifier for object properties.