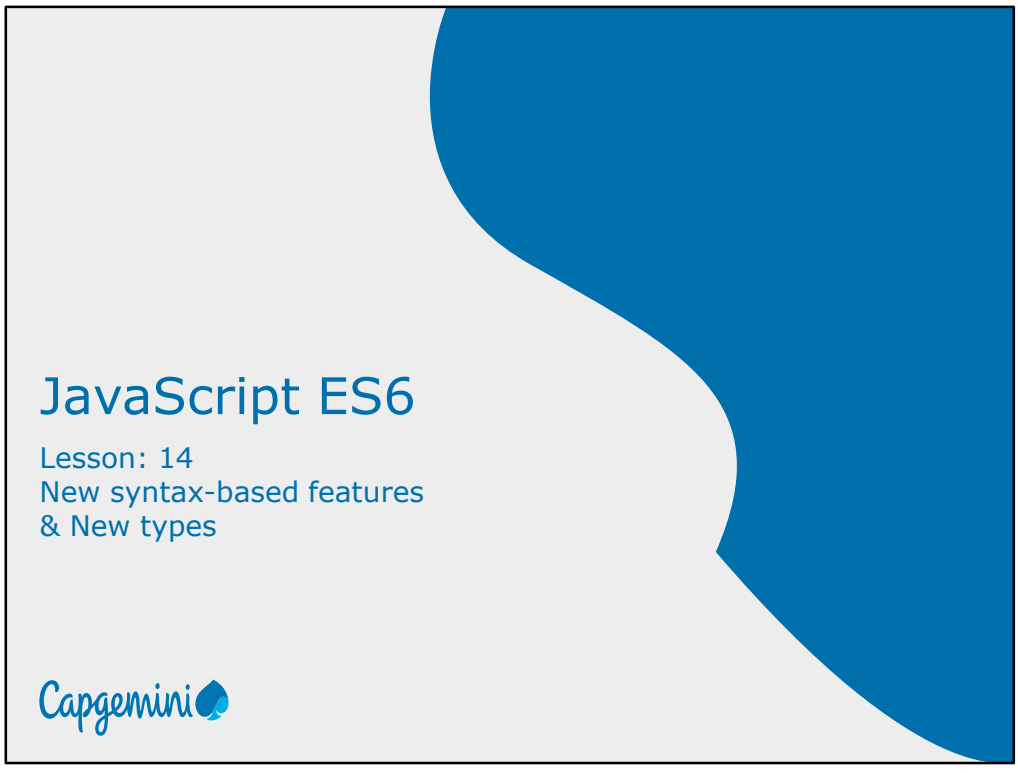
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

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.

Instructor Notes:

Add instructor notes here.

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.

Variables that are declared using the var keyword are called as function scoped variables. Using ***var*** keyword we cannot define block-scoped variables. The function scoped variables are accessible globally to the script, i.e. throughout the script, if declared outside a function. Similarly, if the function scoped variables are declared inside a function, then they become accessible throughout the function, but not outside the function.

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope. i.e. variable can be used before it has been declared. At the same time JavaScript hoists declarations, not initializations.

JavaScript in strict mode does not allow variables to be used if they are not declared. **"use strict";** Defines that JavaScript code should be executed in "strict mode".

```
▶ Uncaught ReferenceError: age is not defined(…)
> let name = "Karthik";
  {
    let name = "Ganesh"; //accesible only in the current block
  }
  console.log("Name : "+name);
  Name : Karthik
< undefined
> let testVar = 1;
  let testVar = 2;
▶ Uncaught SyntaxError: Identifier 'testVar' has already been declared(…)
```

Instructor Notes:

Add instructor notes here.

Demo

let-keyword

Instructor Notes:

Add instructor notes here.

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.

```
> const CONSTANT_VALUE = 30;
  CONSTANT_VALUE = 40
✖ ▶ Uncaught TypeError: Assignment to constant variable.(…)
> const PI = 3.14159265359;
  {
       const PI = 3.14;
       console.log("PI inside the block : "+PI);
  }
  console.log("PI outside the block : "+PI);
  PI inside the block : 3.14
  PI outside the block : 3.14159265359
```

Instructor Notes:

Add instructor notes here.

Demo

const-keyword

Instructor Notes:

Add instructor notes here.

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.

bind(), call() & apply doesn't have any effect on the arrow function.

```
> var foo = {
    number : 54,
    test : function(){
        return () =>console.log(this.number);
    }
}
var baz = {
    number : 20
}

foo.test().bind(baz)();
foo.test().call(baz);
54
54
```

Instructor Notes:

Add instructor notes here.

Demo

arrow-functions

Instructor Notes:

Add instructor notes here.

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.

```
> var generateBaseTax=()=>0.05;
  var generateTotal = function(price, tax = price * generateBaseTax()){
     console.log(price + tax);
  }
  generateTotal(50);
  52.5
> var argumentsPassedTest = function(arg1,arg2 = 20, arg3 = 30){
     console.log(arguments.length);
  }
  argumentsPassedTest(10,undefined);
  2
<- undefined
```

Instructor Notes:

Add instructor notes here.

Demo

default-parameter-values

Instructor Notes:

Add instructor notes here.

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 []

```
> var printCategories = new Function("...categories","return categories;");
  printCategories('printers','mouse pads')
< ["printers", "mouse pads"]
```

Instructor Notes:

Add instructor notes here.

Demo

rest-parameter

Instructor Notes:

Add instructor notes here.

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

```
> var numbers = "456197";
  Math.max(...numbers);
< 9
> var customArray = [..."123",...["4","5","6"]];
  console.log(customArray);
  ["1", "2", "3", "4", "5", "6"]
```

Instructor Notes:

Add instructor notes here.

Demo

spread-operator

Instructor Notes:

Add instructor notes here.

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);
        }
})();
```

Instructor Notes:

Add instructor notes here.

Demo

for…of

Instructor Notes:

Add instructor notes here.

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.

```
> let name = "Karthik";
  console.log(`Name : ${name}`);
  Name : Karthik
> let a = 5;
  let b = 6;
  let result = `Addition of ${a} and ${b} is ${a+b}`;
  console.log(result);
  Addition of 5 and 6 is 11
> /*tag functions*/
  let processInstructions = function(strings,...values){
      console.log(strings);
      console.log(values);
  }

  let id = 714709;
  let department = 'Training'

  processInstructions `Hi ${id} from ${department}.`; //tagged template string
  ▶ ["Hi ", " from ", ".", raw: Array[3]]
  [714709, "Training"]
```

The custom function takes two parameters, that is, the first parameter is an array of string literals of the template string and the second parameter is an array of resolved values of the expressions. The second parameter is passed as multiple arguments therefore we use the rest argument.

Instructor Notes:

Add instructor notes here.

Demo

template-literals

Instructor Notes:

Add instructor notes here.

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.

```
> let myArray = [1, 2, 3];
  let [a, b, c] =  myArray;
  console.log("a = %d b = %d c = %d",a,b,c);
  a = 1 b = 2 c = 3
> let products = ["Television","Mobile Phone","Music Player"];
  let productArray = [tv,...others] = products;
  console.log(tv,others);
  Television ["Mobile Phone", "Music Player"]
> let verifySalary=function([low,average],high){
      console.log(average);
  }
  verifySalary([10000,150000],50000);
  150000
> let employee = {
      id:714709,
      name:'Karthik'
  }
  let {id,name} = employee;
  console.log(id,name);
  714709 "Karthik"
> let person = {
      first:'Karthik',
      last:'Muthukrishnan'
  }
  let {first:firstName,last:lastName} = person;
  console.log(firstName,lastName);
  Karthik Muthukrishnan
```

Instructor Notes:

Add instructor notes here.

Demo

destructuring

Instructor Notes:

Add instructor notes here.

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.

Instructor Notes:

Add instructor notes here.

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).

Instructor Notes:

Add instructor notes here.

Demo

symbol

Instructor Notes:

Add instructor notes here.

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

While referring to the well-known symbols in the text, we usually prefix them using the @@ notation. For example, the Symbol.iterator symbol is referred to as the @@iterator method. This is done to make it easier to refer to the well-known symbols in the text.

Instructor Notes:

Add instructor notes here.

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.