ES6(ECMAScript)

Presentation By :Mahesh Ravirala

Agenda

- Overview
- Features
- Usage/Examples
- References

Brendan Eich, a developer at Netscape Communications Corporation, in 1995

Before and After

Features

- Syntax
- Destructuring
- Modules
- Classes
- Arrow Functions
- Promises
- Collections
- Array extensions
- Object extensions
- String extensions
- Symbol
- Iterators & Generators

1.Syntax

let	• block-scoped variables using the let keyword • let name='impact'
const	• constants using the const keyword • const baseURL = 'https://citiustech.com/'
var	• functional scope • var x=100;

let vs var vs const

let

- Block scope
- Not hoisted
- Mutable
- Redeclaration not possible

var

- Functional scope
- Hoisted
- Mutable
- Redeclaration possible

const

- block scope
- Not hoisted
- Immutable
- Redeclaration not possible

```
//Scope : let is block scope
if(true){
  let y=3;
}
console.log(y) // output : ReferenceError


// Redeclaration
let counter = 10;
let counter; // error

//Hoisting
console.log(x); // ReferenceError
let x=10;
console.log(x); // 10

/mutable
let name='Mahesh';
name ="mahesh netha"
console.log(name); // mahesh netha
```

```
//Scope : var is functional scope
if(true){
  var x=3;
}
console.log(x) // output : 3

// Redeclaration
var counter = 10;
var counter;
console.log(counter); // 10

//Hoisting
console.log(x); // undefined
var x=10;
console.log(x); // 10
//mutable
var name='Mahesh';
name ="mahesh netha"
console.log(name); // mahesh netha
```

```
//Scope : const is block scope
if(true){
 const y=3;
}
console.log(y) // output : ReferenceError


// Redeclaration
const  domain = 'citiustech';
const domain = "citiustech.com"; // Identifier 'domain' has already been declared

//Hoisting
console.log(x); // ReferenceError
const x=10;
console.log(x); // 10

//immutable
const  domain = 'citiustech';
domain = "citiustech.com";
console.log(domain) // TypeError: Assignment to constant variable.
```

Default Parameters

- function fn(param1=default1, param2=default2,..) { }
- function say(message='Hi') { console.log(message); } say(); // *'Hi'*

Rest Parameter

- (...)- packs elements into an array
- function fn(a,b,...args) { //... }
- fn(1, 2, 3, "A", "B", "C"); args = [3,'A','B','C']
- function fn(a,...rest, b) { // *error* } …rest should be last

Spread Operator

- (…) , spreads/unpacks elements
- const odd = [1,3,5]; const combined = [2,4,6, ...odd]; console.log(combined);
- // [2, 4, 6, 1, 3, 5]
- Array ,object copy and concat

Object Literal

- Simplicity , shorthand syntax (concise)
- let name = 'Computer', status = 'On';
- let machine = { name, status }
- Before ES6 : machine = {name:name,status:status }

Template Literal

- single quotes (‘) or double quotes (“) -limited functionality
- by wrapping your text in backticks (``)
- Feature – Multiline, String formatting,HTML Escaping
- let firstName = 'John', lastName = 'Doe'; let greeting = `Hi ${firstName}, ${lastName}`;

for..of

- for (variable of iterable) { // *statements* }
- let colors = ['Red', 'Green', 'Blue'];

- for (const [index, color] of colors.entries()) {
- console.log(`${color} is at index ${index}`);}

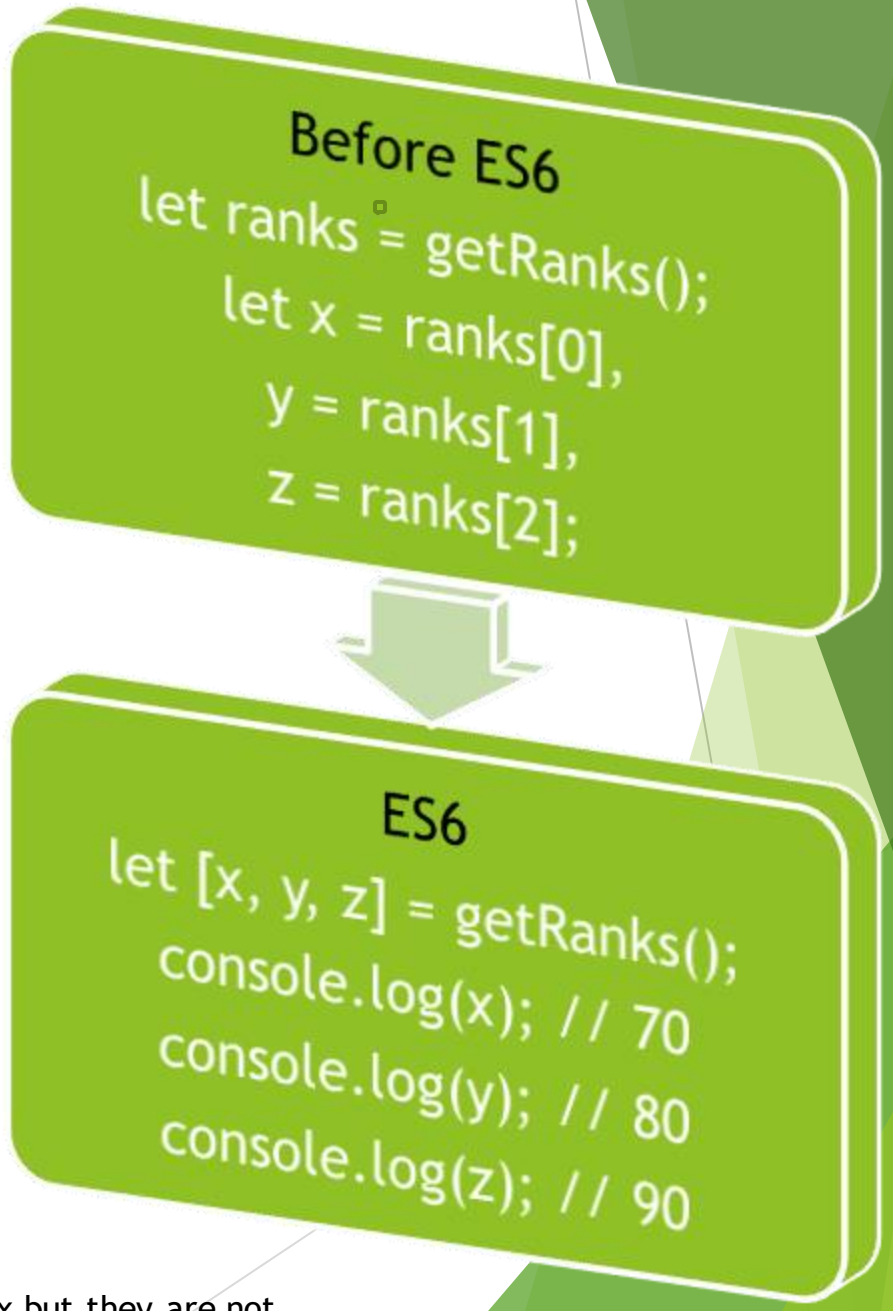
2.De-structuring

2.1 Array : [x,y,z,….]

ES6 de-structuring assignment that allows you to de-structure an array into individual variables.

Example:

```
const [a,b] = [1,2]
console.log(a) // 1
function getRanks() {
  return [1, 2, 3];
}
```



Note

Note that the square brackets [] look like the array syntax but they are not.

More

```
/*If the getScores() function returns an array of two elements,
 the third variable will be undefined, like this: */
function getMarks() {
    return [70, 80];
}
let [x, y, z] = getMarks();
console.log(x); // 70
console.log(y); // 80
console.log(z); // undefined

/* In case the getMarks() function returns an array that has more than
 three elements, the remaining elements are discarded. For example*/
function getMarks() {
    return [70, 80, 90, 100];
}
let [x, y, z] = getMarks();
console.log(x); // 70
console.log(y); // 80
console.log(z); // 90
//Array Destructuring Assignment and Rest syntax
/* take all remaining elements of an array and put them in a new
array by using the rest syntax (...):*/
let [x, y ,...args] = getMarks();
console.log(x); // 70
console.log(y); // 80
console.log(args); // [90, 100]
//Default values
let a, b;
[a = 1, b = 2] = [10];
console.log(a); // 10
console.log(b); // 2
```

2.De-structuring

2.2 Object : {x,y}

ES6 de-structuring assignment that allows you to de-structure an array into individual variables.

Syntax :

```
let { property1: variable1,
property2: variable2 } = object;
```

Example:

```
let player= {
        name:”Jarvo”,
        jersey:69 };
```

Note

property name is always on the left whether it’s an object literal or object destructuring syntax.

More

```
/* If the variables have the same names as the properties of the object,
you can make the code more concise as follows:*/

let { firstName, lastName } = person;

console.log(firstName); // 'John'
console.log(lastName); // 'cena'
/* When you assign a property that does not exist to a variable using the object
destructuring, the variable is set to undefined. For example:
*/

let { firstName, lastName, middleName } = person;
console.log(middleName); // undefined

//default value
let person = {
    firstName: 'John',
    lastName: 'cena',
    currentAge: 28
};

let { firstName, lastName, middleName = '', currentAge: age = 18 } = person;

console.log(middleName); // ''
console.log(age); // 28

//Destructuring a null object
function getPerson() {
    return null;
}
let { firstName, lastName } = getPerson() || {};
```

3.Modules

- A module is nothing more than a chunk of JavaScript code written in a file
- **ES6 modules** and how to export variables, functions, classes from a module, and reuse them in other modules.
- By default, variables and functions of a module are not available for use
- export keyword can be used to export components in a module
- Ex: patient.js , admin.js , physician.js
- export and import keywords

Modules in ES6 work only in **strict mode**

Exporting a Module

- export keyword can be used to export components in a module
- Exports in a module can be classified as follows −

1.Named Exports

- Named exports are distinguished by their names
- several named exports in a module
- export componet1 ; export component2;
- using a single export keyword with {}
- export { componet1,componet2…}

2. Default Exports

- Modules that need to export only a single value can use default exports
- There can be only one default export per module.
- export default component_name

Importing a Module

- import keyword can be used to import components from a module
- A module can have multiple **import statements**.

```
Import { compx as comp1
,compy as comp2} from
module_a;

Import * as var_name from
m_name;
```

1. Importing Named Exports

- While importing named exports, the names of the corresponding components must match.
- import {comp1,comp2} from module_name

2. Default Exports

- Unlike named exports, a default export can be imported with any name.
- Import var_name from module_name

Examples Named Exports

```
let patient = "xyz"

let getPatient = function(){
    return patient.toUpperCase()
}

let setPatient = function(newValue){
    patient = newValue
}

export {patient,getPatient,setPatient}
```

patient.js

```
//Approach 1
import {patient,getPatient} from './patient.js'
console.log(patient)
console.log(getPatient())

//Approach 2
import {patient as x, getPatient as y} from './patient.js'
console.log(x)
console.log(y())

//Approach 3
import * as patient from './patient.js'
console.log(patient.getPatient())
console.log(patient.setPatient('abc'))
```

Util.js

```
let name = 'xyz'

let payer = {
    getName:function(){
        return name
    },
    setName:function(newName){
        name = newName
    }
}

export default payer
```

payer.js

```
import p from './payer.js'
console.log(p.getName())
c.setName('abc')
console.log(p.getName())
```

Util.js

Examples default Exports

- Limitation : must use import import/export outside other statements and functions

```
if( requiredSum ) {
      export sum;          //Syntax error
}
   function importSum() {
     import {sum} from './cal.js';  //Syntax error
       }
```

Dynamic Import:

```
function eventHandler() {
   import('./module1.js')
     .then((ns) => {
        // use the module
        ns.func();
     })
     .catch((error) => {
        // handle error
     });
}
```

(4)=>{ Arrow functions }

- alternative way to write a shorter syntax (concise)compared to the function expression.
- Syntax

(p1, p2, ..., pn) => expression;

```
let add = function (x, y) {
        return x + y;                    // normal function expression
};
console.log(add(10, 20)); // 30

let add = (x, y) => x + y;              // arrow function
console.log(add(10, 20)); // 30;
```

->Single parameter : let inc = x => x*1;
->2Parameters : let inc = (x,y)=> x+y;

->no patameter : let inc = ()=> 7
->more then one statement : let inc=(x) { console.log(x*x); return x*x;}

Arrow vs regular functions

()=>{}

- Syntax
- No arguments binding-showArgs : () => {
  console.log(...arguments);
  } // arguments not defined
- arrow functions do not have their own this.
- not constructible
- never have duplicate named parameters

function(){}

- syntax
- Arguments bindin:-showArgs(){
  console.log(arguments);
  }
- Has
- Constructible
- Can have, however, when using strict mode it cant have:

References

- https://www.javascripttutorial.net/es6/
- https://www.tutorialspoint.com/es6/index.htm
- https://www.slideshare.net/hesher/es2015-es6-overview

```
const  to = ‘CTzen’
let msg =  `Thank you
            ${to}!`
const say=(msg)=> console.log(msg)
say(msg);
```

Thank you CTZen