ES6

Presented by : Pankaj Sharma

Topics to be covered

- Introduction to ES6
    - Transpillers
    - Hands on
    - Strict mode
    - Hoisting
    - Variable
    - Object
    - Class
    - Arrow function

ES6

ES6/ ECMASCRIT6/ ECMASCRIPT 2015

- Classes
- Constants
- Arrow function
- Generators

Let’s dive into ES6

ES6 - Transpillers

Problem if we use ES6:
All browsers are not updated to process the ES6 code.

Solution or Purpose of Transpillers:
A Transpiller takes the ES6 Source code and generates ES5 code that can run in every browser

There are two alternatives to transpile the ES6 code.

1. **Traceur** : It is a Google project

2. **BabelJS** : A Project started by young developer, Sebastian McKenzie (He was 17 Years old that time)
   https://babeljs.io/

ES6 - HandsOn

Create below code in any text editor (preferably in Visual Studio / ATOM)

```
var message="Hello World";
console.log(message);
```

1. Save it with test.js file name
2. Goto folder location where you saved this file.
3. Open folder in Command propmt
4. Run below command

```
node test.js
```

Note: node js should be installed on your system.

You will get the out as “Hello World”

Do a small change in your application, remove the var keyword, now save and rerun the node test.js command.

ES6 – Strict Mode

The fifth edition of ECMAScript specification introduces the Strict Mode.

```
"use strict"
message="Hello World";
console.log(message);
```

1. Save the changes and rerun the node test.js command.

Output

```
message="Hello World";
     ^
ReferenceError: message is not defined
```

You can also use the strict mode inside scope , like function or if block etc.it will make impose the strict mode inside the block.

ES6 – ES6 and Hoisting

The JavaScript engine, by default, moves declarations to the top. This feature is termed as **hoisting**. This feature applies to variables and functions. Hoisting allows JavaScript to use a component before it has been declared. However, the concept of hoisting does not apply to scripts that are run in the Strict Mode.

ES6 – Variable

Variable is a "named space in memory." It works as a container to hold values in program.

```
//ES5 syntax for declaring a variable
var variable_name;
```

ES6 introduces following variable declaration syntax

- Using let
- Using const

ES6 – Variable – JavaScript and Dynamic Typeing

JavaScript is an un-typed language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically. This feature is termed as **dynamic typing**.

```
var message="Hello World";
message=10;
console.log(message);
```

While executing this code it will not show any error, rather it will assign new value type in message and message will be displayed.

ES6 – Variable – JavaScript variable scope

JavaScript defines only two scopes.

- Global Scope: a variable with global scope can be accessed from any part of the JS code
- Local Scope: a variable with local scope can be accessed within a function where it is declared.

```
var num=10;
function test(){
var num=100;
console.log("value of num in test()"+num);
}
console.log("value of num outside test()"+num);

test()
```

What will be the output of above code

ES6 – Variable – JavaScript variable scope

```
var a=10;
var a=20;
console.log(a);
```

What will be the output of above code

ES6 – Variable – JavaScript const

```
const a=10;
a=20;
console.log(a);
```

What will be the output of above code

Rules about constant:

- Constant cannot be reassigned a value
- Constant cannot be declared
- A Constant requires initialization
- A value assigned to constant value is mutable

ES6 – Variable – JavaScript Hoisting

Is it possible to access a variable outside its scope?

```
function main(){
for(var i=1;i<=5;i++){
console.log(i);
}

console.log("value of i = "+i);
}

main();
```

What will be the output of above code

i is declared inside the for() but still available outside the for(), this is variable hoisting

ES6 – Objects

An **object** is an instance which contains a set of key value pairs. Unlike primitive data types, objects can represent multiple or complex values and can change over their life time. The values can be scalar values or functions or even array of other objects.

Object Declaration Syntax

```
var identifier = {
Key1:value,
Key2: function () {
//functions },

Key3: [“content1”,” content2”]
}
```

Syntax of accessing object’s property

```
objectName.property
```

ES6 – Objects

Write the below code in text editor, and execute it

```
var employee={
name: "pankaj sharma",
salary: 90000,
showDetail: function(){
console.log("Name : "+this.name);
console.log("Salary : "+this.salary);
}
}

employee.showDetail();
```

Here employee is an object having name, salary and showDetail as properties.

ES6 – Objects-constructor

JavaScript provides a special constructor function called **Object()** to build the object. The new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method.

What will be the output of below code?

```
var testObj=new Object();
console.log(testObj);
```

Output : {}

Now if you want to add some property along with value to be associated. Then we use the below methods.

```
testObj.property=value;
Or
testObj[“key”]=value
```

ES6 – Objects-constructor

Check for below code:

```
var car=new Object();
car.make="Maruti";
car.model="Maruti Alto LX";
car.year=2008;
console.log(car);
```

ES6 – Objects-constructor

One more way of creating object: using function

```
function Car(){
this.make="Maruti"
this.model="Maruti Alto LX"
this.year=2008
}

var car=new Car();
console.log(car);
```

```
function Car(){
this.make="Maruti"
}

var car=new Car();
car.model="Maruti Alto Lx";
console.log(car);
```

ES6 – Objects (Creating Object using Object.create() method)

Objects can also be created using the **Object.create()** method. It allows you to create the prototype for the object you want, without having to define a constructor function.

```
var roles={
type:"Admin", // default role
showType: function(){
// show the type of role
console.log(this.type);
}
}

//create new super_role
var super_role=Object.create(roles);
super_role.showType();
```

Make changes to above code to create new roles as “Manager”

ES6 – Objects (Creating Object using Object.create() method)

Solution

```
var roles={
type:"Admin", // default role
showType: function(){
// show the type of role
console.log(this.type);
}
}

//create new super_role
var super_role=Object.create(roles);
super_role.showType();

//Create new manager_role
var manager_role=Object.create(roles);
manager_role.type="Manager";
manager_role.showType();
```

ES6 – Objects (Object.assign() function)

The **Object.assign()** method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object.

Syntax
Object.assign(target,…sources);

Object cloning

```
var employee=new Object();
employee.name="Pankaj";
employee.salary=90000;

var employeeCopy=Object.assign({},employee);
console.log("employee: ",employee);
console.log("employeeCopy: ",employeeCopy);
employeeCopy.id=1004482;
console.log("employeeCopy: ",employeeCopy);
console.log("employee: ",employee);
```

ES6 – Class

Write below code:

```
class Employee{
constructor(name,salary){
this.name=name;
this.salary=salary;
}

showEmployeeDetail(){
console.log('name : ',this.name);
console.log('salary : ',this.salary);
}
}

var employee=new Employee('pankaj',90000);
employee.showEmployeeDetail();
```

Thank You!

Email: info@yash.com
Web: www.yash.com

Presented by : Pankaj Sharma