ES6 Introduction

•Introduction to React.JS

•Working with Components

•Props and State

•Component Life Cycle

•Forms

•Working with Lists and Conditionals

•Working with AJAX

•Introduction to routing

•Debugging React Apps

•Overview of Hooks

•State Management using Redux

ES6 Features

•let and const keyword

•Arrow functions

•‘this’ keyword in JavaScript

•Default Function parameters

•Rest and Spread Operators

•Template Literals

•Object and Array De-structuring

•Modular Programming

•Default and Named Exports

let keyword

•The let statement allows you to declare a variable with block scope.

•let keyword cannot be hoisted

.•Variables declared with the var keyword can not have Block Scope.

•In case of let keyword, variables declared inside a block {} can not be accessed from outside the block.

let keyword : Example1

functiongreetPerson2(name){

letgreet;

if(name==='John'){

greet="HelloJohn";//This greet variable become blocked scoped.

}else{

greet="Hithere";

}console.log(greet);//greet has a functional scope.

let greet;//let declaration cannot be hoisted.

}

greetPerson2("John");

let keyword : Example 2 -Scope

var a=1;

var b=2; if(a===1){

var a=10;

let b=20;

console.log(a);

console.log(b);

}

console.log(a);

console.log(b);

let keyword : Example 3 -Redeclaration

var a =10;

var a = 2;//We can redeclare

let d =1;

let d =3;//We cannot redeclare

let keyword : Example 4 -In Async Method call

let keyword best suites for Async Methods like setTimeout.

For(let i=1;i<=10;i++){

setTimeout(function(){console.log(i)},1000);

}

const keyword

•The const statement allows you to declare a constant (a JavaScript variable with a constant value).

•Constants are like let variables, except that the value cannot be changed.

•JavaScript const variables must be assigned a value when they are declared.

•The keyword const is a little misleading.

•It does NOT define a constant value. It defines a constant reference to a value.

•Because of this, we cannot change constant primitive values, but we can change the properties of constant objects.

Const keyword

let num1;

constnum2=20;//const declaration must be initialized.

const obj1={

name: “Anup"

};

console.log(obj1.name);

//obj1={};//Doesnotwork

obj1.name='Chandler';

console.log(obj1.name);

let vs const keyword

const PI=3.14;//To declare Universal constants

const MAX\_SIZE=100;//To declare array size

let a=5;

let b=10;

a= a+b;//We are reassigning variables..

b=a-b;

a=a-b;

console.log(a);//10

console.log(b);//5

Arrow function / expression

* Arrow functions allow us to write shorter syntax

//Without Arrow Function

hello = function(){

return “Hello World!”

}

//With Arrow Function

hello=()=>{

return “Hello World!”

}

* It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword

hello=()=>”Hello World”

Arrow function / expression-Examples

Var getRegularValue = function(){

return 10;

}

console.log(getRegularvalue);

//How to convert this function to arrow function

const getArrowValue = ()=>{

return 10;

}

Lexical “this” keyword

* The handling of “this” is also different in arrow functions compared to regular functions. In short with arrow functions there are no binding of “this”.
* In regular functions “this” keyword represented the object that called the function, which could be the window, the document, a button or whatever.
* With arrow functions “this” keyword always represents the object that defined the arrow function.

Lexical “this” keyword-Example1

//Without Arrow Function

var employee = {

id:1;

greet: function(){

var self = this;

setTimeout(function(){console.log(self.id)},1000);

}

}

employee.greet();

//With Arrow Function

var employee = {

id:1;

greet: function(){

setTimeout(function()=>{console.log(this.id)},1000)

}

}

employee.greet();

Default Function parameters

•If a function is called with missing arguments (less than declared), the missing values are set to undefined.

•Sometimes this is acceptable, but sometimes it is better to assign a default value to the parameter.

•ECMAScript 2015 allows default parameter values in the function declaration

function (a=1, b=1) {

// function code

}

Default Function parameters-Example1

let getValue = function(value){

console.log(value)

}

getValue(); //This will throw error

//With Default function parameters

let getValue = function(value=10){

console.log(value)

}

getValue(); //This will not throw error

Default Function parameters-Example2

let percentBonus=()=>0.1;

let getValue = function(value=10,bonus=value\*percentBonus()){

console.log(value+bonus);

console.log(arguments.length);

}

getValue();//11,0

getValue(20);//22,1

getValue(20,30);//50,2

getValue(undefined,30);//40,2

Rest Operator

Aggregation of remaining arguments into single parameter of functions.

function f (x, y, ...a) {

return (x + y) \* a.length

}

f(1, 2, "hello", true, 7) === 9

Rest Operator-Example