Main Points/Key Points	Notes

	ES6 Features
	
	1. Undefined vs. ReferenceError.
	a. Any **<u>undeclared variable</u>** will be assigned to ***undefined*** at execution and contains ***undefined*** data type.
	b. Any **<u>declared variable</u>** without any value assignment contains ***undefined*** data type;
	c. ***ReferenceError*** is thrown when you are trying to access **<u>undeclared variable</u>**.
	d. Example:
	
	Code
	`console.log(typeof myoutput);`
	`console.log(myoutput);`
	
	JavaScript Engine
	`var myoutput;`
	`console.log(typeof myoutput);`
	`console.log(myoutput);`
	
	e. Both of ***undefined*** and ***ReferenceError*** are having different behavior when it comes to hoisting.
	
	2. **Java vs. JavaScript Data Types.**
	
	Java
	1. Boolean
	2. Byte
	3. Char
	4. Short
	5. Int
	6. Long
	7. Float
	8. Double
	
	JavaScript
	1. Boolean
	2. Null
	3. Undefined
	4. Number
	5. String
	6. Symbol
	7. Object

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	3. **Hoisting Variables**.
	a. The way variables are ***declared*** and ***initialized*** in JavaScript.
	b. Example:
	
	Code
	`var a = 200;`
	JavaScript Engine
	`var a;`
	`a = 200;`
	
	c. In JavaScript, every variable declared with *var* keyword will be placed on top of the scope.
	d. Example:
	
	Code
	`console.log(output);`
	`var output = "This is my output";`
	JavaScript Engine
	`var output;`
	`console.log(output);`
	`output = “This is my output”;`
	
	e. Variable declarations are processed before code execution. This includes variable initializations.

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	4. **Global vs. Local vs. Block Scope.**
	a. Any **<u>undeclared variable</u>** is a ***global variable*** and placed inside a **global scope**.
	b. Example:
	
	Code
	`function hoist(){`
	`  x = 10;`
	`  var y = 101;`
	`}`
	
	`hoist();`
	`console.log(x);`
	`console.log(y);`
	
	JavaScript Engine
	`var x;`
	`function hoist(){`
	`  x = 10;`
	`  var y = 101;`
	`}`
	
	`hoist();`
	`console.log(x);`
	`console.log(y);`

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	c. Any **declared variable** within a function is a ***local variable*** and placed inside a **local scope**.
	d. Example:
	
	Code
	`function output(){`
	`  console.log(x);`
	`  var x = "This is my output";`
	`}`
	
	`output();`
	
	JavaScript Engine
	`function output(){`
	`  var x;`
	`  console.log(x);`
	`  x = "This is my output";`
	`}`
	
	`output();`

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	e. Any variable with ***let*** or ***const*** keyword is placed inside a block scope; they can be **global** or **local variable**.
	
	Code
	`console.log(output);`
	`let output = "This is my output";`
	
	`let x;`
	`console.log(x);`
	`x = "This is my second output";`
	
	JavaScript Engine
	`console.log(output);`
	`let output;`
	`output = "This is my output";`
	
	`let x;`
	`console.log(x);`
	`x = "This is my second output";`
	
	5. **Strict Mode.**
	a. Any **<u>undeclared variable</u>** is restricted to avoid ***undefined*** and ***redefined*** values.
	b. It eliminates explicit ***JavaScript errors*** by throwing an error or exception to the developers.

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	c. It fixes difficult mistakes for JavaScript Engine to perform ***optimization***.
	d. Prevent ***experimental JavaScript*** to be included in the current code.
	e. Example:
	
	Code
	`"use strict";`
	
	`console.log(output);`
	`output = "This is my output";`
	
	JavaScript Engine
	`"use strict";`
	
	`console.log(output);`
	`output = "This is my output";`
	
	`// variable output is not hoisted.`
	`// treated as syntax error.`

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	6. **var vs. let vs. const.**
	a. Any **declared variable** with ***var*** keyword is hoisted and it can be ***re-declared*** and ***updated***. This creates a logic error if it is not properly observed.
	b. Example:
	
	Code
	`var salam = "Salam";`
	`var times = 4;`
	
	`if (times > 3) {`
	`  var salam = "say Hello instead";`
	`}`
	
	`console.log(salam);`
	`//"say Hello instead"`
	`// salam is re-declared and updated`
	
	c. A variable declared with ***let*** keyword is placed in a block scope and ***NOT initialized*** (*ReferenceError*).

	Summary

Main Points/Key Points	Notes
	ES6 Features d. A variable declared with ***let*** keyword can be ***updated*** but ***NOT re-declared***. e. Example:
	Summary

Code

`let salam = "say Salam";`
`salam = "say Hello instead";`
`// Updated to 'say Hello instead'`

`let salam = "say Salam";`
`let salam = "say Hello instead";`
`//error: Identifier 'salam' has already been declared`

`let salam = "say Salam";`
`if (true) {`
`let salam = "say Hello instead";`
`console.log(salam);`
`//"say Hello instead"`
`}`
`console.log(salam);`

`console.log(output);`
`let output = "This is my output";`
`let x;`
`console.log(x);`
`x = "This is my second output";`

Main Points/Key Points	Notes

	ES6 Features
	
	f. A variable declared with ***const*** keyword is placed in a block scope and ***CANNOT be changed*** (immutable).
	g. It is hoisted but ***NEED to be initialized*** with a value. (*ReferenceError*).
	h. A variable declared with ***const*** keyword ***CANNOT be updated or re-declared.***
	i. Example:
	
	Code
	
	`const salam = "say Salam";`
	`salam = "say Hello instead";`
	`//error: Assignment to constant variable.`
	
	`const salam = "say Salam";`
	`const salam = "say Hello instead";`
	`//error: Identifier 'salam' has already been declared`
	
	`const cgpa = 3.5;`
	`cgpa = 3.5 * 2;`
	`console.log(cgpa);`
	
	`const pi;`
	`console.log(pi);`
	`pi = 3.142;`

	Summary

Main Points/Key Points	Notes

	ES6 Features
	
	j. However, any object or array declared with ***const*** keyword, the property or element can be **ADDED**.
	k. The existing property or element **CANNOT** be updated.
	l. Example:
	
	Code
	
	`const car = {`
	`  manufacturer: 'Honda',`
	`  year: '2000',`
	`  mileage: '15000'`
	`}`
	
	`car.owner = 'Ali';`
	`console.log(car);`
	
	`const colors = ['blue','red','green'];`
	
	`colors.push('yellow');`
	`console.log(colors);`
	
	m. Overall summary for ***var***, ***let*** and ***const*** keywords.

	var	**let**	**const**
Scope	Global/ Local	Block	Block
Declare	Yes	Yes	Yes with Initialization
Initialization	Yes (Undefined)	No	No
Update	Yes	Yes	No
Re-Declare	Yes	No	No

Summary

Main Points/Key Points	Notes
	ES6 Features **7. Hoisting Functions and Classes.** a. A function can be declared in two ways: i. Declarations - **`function () {…}`** ii. Expressions - **`var x = function(){…};`** b. Function declarations are hoisted to the top of the scope. However, function expressions are **NOT** hoisted. c. Example:

Function Declarations

```
hoisted();
function hoisted(){
 console.log("My Output!");
}
// Display "My Output!"
```

Function Expressions

```
hoisted();
var hoisted = function(){
 console.log("My Output!");
};

//TypeError: expression is not a
function
```

Declaration + Expressions

```
expression();
var expression = function hoisted(){
 console.log("My Output!");
};

//TypeError: expression is not a
function
```

Summary

Main Points/Key Points	Notes
	ES6 Features d. A class can be declared in two ways: i. Declarations – **`class car () {…}`** ii. Expressions – **`var car = class (){…};`** e. Class declarations are hoisted but remain uninitialized until evaluation. However, class expressions are **NOT** hoisted. f. A class has to be declared before you can use it. g. Example:

Class Declarations

```
var square = new Polygon();
square.height = 10;
square.width = 10;
console.log(square);
// TypeError: Polygon is not a
constructor

class Polygon {
  constructor(height, width){
  this.height = height;
  this.width = width;
 }
}
```

Class Expressions

```
var square = new Polygon();
square.height = 10;
square.width = 10;
console.log(square);
// TypeError: Polygon is not a
constructor

var Polygon = class Polygon {
  constructor(height, width){
  this.height = height;
  this.width = width;
 }
};
```

Summary

Main Points/Key Points	Notes
	ES6 Features **8. References:** a. Mabishi, E. (2017). *Understanding Hoisting in JavaScript*. Retrieved from https://scotch.io/tutorials/understanding-hoisting-in-javascript b. Chima, S. (2017). Var, let, const – what's the difference. Retrieved from https://dev.to/sarah_chima/var-let-and-const--whats-the-difference-69e c. ES6 Tutorial (2018). Retrieved from https://www.tutorialspoint.com/es6/index.htm
	Summary