**RnD Day 6**

**Syntax Error:**

An [exception](https://developer.mozilla.org/en-US/docs/Glossary/Exception) caused by the incorrect use of a pre-defined [syntax](https://developer.mozilla.org/en-US/docs/Glossary/Syntax). Syntax errors are detected while compiling or parsing source code.

****Runtime Error:****

A runtime error is an error that occurs during the running of the program, also known as the exceptions. In the example that is given below the syntax is correct, but at runtime, it is trying to call a method that does not exist.

**Example:**

**<script type="text/javascript">**

**window.printme(); // An runtime error here**

**</script>**

**Logic Errors:**

These are errors where the syntax is actually correct but the code is not what you intended it to be, meaning that program runs successfully but gives incorrect results.

**ES6 Features:**

**1. let and const keywords :**

The keyword "let" enables the users to define variables and on the other hand, "const" enables the users to define constants. Variables were previously declared using "var" which had function scope and were hoisted to the top. It means that a variable can be used before declaration. But, the "let" variables and constants have block scope which is surrounded by curly-braces "{}" and cannot be used before declaration.

*let i = 10;*

*console.log(i); //Output 10*

*const PI = 3.14;*

*console.log(PI); //Output 3.14*

**2. Arrow Functions:**

ES6 provides feature known as Arrow Functions. It provides a more concise syntax for writing function expressions by removing out the "function" and "return" keywords.  
 Arrow functions are defined using the fat arrow (=>) notation.

*let sumOfTwoNumbers = (a, b) => a + b;*

*console.log(sum(10, 20)); // Output 30*

It is evident that, there is no "return" or "function" keyword in the arrow function declaration.

**3. Multi-line Strings:**

ES6 also provides Multi-line Strings. Users can create multi-line strings by using back-ticks(`).  
It can be done as shown below :

*let greeting = `Hello World,*

*Greetings to all,*

*Keep Learning and Practicing!`*

**4. Default Parameters:**

In ES6, users can provide the default values right in the signature of the functions. But, in ES5, OR operator had to be used.

*let calculateArea = function(height = 100, width = 50) {*

*// logic*

*}*

**5. Template Literals:**

ES6 introduces very simple string templates along with placeholders for the variables. The syntax for using the string template is ${PARAMETER} and is used inside of the back-ticked string.

*let name = `My name is ${firstName} ${lastName}`*

**6. Destructuring Assignment:**

Destructuring is one of the most popular features of ES6. The destructuring assignment is an expression that makes it easy to extract values from arrays, or properties from objects, into distinct variables.  
 There are two types of destructuring assignment expressions, namely, Array Destructuring and Object Destructuring. It can be used in the following manner :

//Array Destructuring

*let fruits = ["Apple", "Banana"];*

*let [a, b] = fruits; // Array destructuring assignment*

*console.log(a, b);*

//Object Destructuring

*let person = {name: "Peter", age: 28};*

*let {name, age} = person; // Object destructuring assignment*

*console.log(name, age);*

**7. Enhanced Object Literals:**

ES6 provides enhanced object literals which make it easy to quickly create objects with properties inside the curly braces.

*function getMobile(manufacturer, model, year) {*

*return {*

*manufacturer,*

*model,*

*year*

*}*

*}*

*getMobile("Samsung", "Galaxy", "2020");*

**8. Promises:**

In ES6, Promises are used for asynchronous execution. We can use promise with the arrow function as demonstrated below.

*var asyncCall = new Promise((resolve, reject) => {*

*// do something*

*resolve();*

*}).then(()=> {*

*console.log('DON!');*

*})*

**9. Classes:**

Previously, classes never existed in JavaScript. Classes are introduced in ES6 which looks similar to classes in other object-oriented languages, such as C++, Java, PHP, etc. But, they do not work exactly the same way. ES6 classes make it simpler to create objects, implement inheritance by using the "extends" keyword and also reuse the code efficiently.  
 In ES6, we can declare a class using the new "class" keyword followed by the name of the class.

*class UserProfile {*

*constructor(firstName, lastName) {*

*this.firstName = firstName;*

*this.lastName = lastName;*

*}*

*getName() {*

*console.log(`The Full-Name is ${this.firstName} ${this.lastName}`);*

*}*

*}*

*let obj = new UserProfile('John', 'Smith');*

*obj.getName();* // output: The Full-Name is John Smith

**10. Modules:**

Previously, there was no native support for modules in JavaScript. ES6 introduced a new feature called modules, in which each module is represented by a separate ".js" file. We can use the "import" or "export" statement in a module to import or export variables, functions, classes or any other component from/to different files and modules.

*export var num = 50;*

*export function getName(fullName) {*

*//data*

*};*

*import {num, getName} from 'module';*

*console.log(num); // 50*