ES6 - Syntax

Syntax defines the set of rules for writing programs. Every language specification defines its own syntax.

A JavaScript program can be composed of −

- **Variables** − Represents a named memory block that can store values for the program.
- **Literals** − Represents constant/fixed values.
- **Operators** − Symbols that define how the operands will be processed.
- **Keywords** − Words that have a special meaning in the context of a language.

The following table lists some keywords in JavaScript. Some commonly used keywords are listed in the following table.

break	as	any	Switch
case	if	throw	Else
var	number	string	Get
module	type	instanceof	Typeof
finally	for	enum	Export
while	void	this	New
null	super	Catch	let
static	return	True	False

- **Modules** − Represents code blocks that can be reused across different programs/scripts.
- **Comments** − Used to improve code readability. These are ignored by the JavaScript engine.
- **Identifiers** − These are the names given to elements in a program like variables, functions, etc. The rules for identifiers are −

- Identifiers can include both, characters and digits. However, the identifier cannot begin with a digit.
- Identifiers cannot include special symbols except for underscore (_) or a dollar sign ($).
- Identifiers cannot be keywords. They must be unique.
- Identifiers are case sensitive. Identifiers cannot contain spaces.

The following table illustrates some valid and invalid identifiers.

Examples of valid identifiers	Examples of invalid identifiers
firstName	Var#
first_name	first name
num1	first-name
$result	1number

Whitespace and Line Breaks

ES6 ignores spaces, tabs, and newlines that appear in programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

JavaScript is Case-sensitive

JavaScript is case-sensitive. This means that JavaScript differentiates between the uppercase and the lowercase characters.

Semicolons are Optional

Each line of instruction is called a **statement**. Semicolons are optional in JavaScript.

Example

```
console.log("hello world")
console.log("We are learning ES6")
```

A single line can contain multiple statements. However, these statements must be separated by a semicolon.

Comments in JavaScript

Comments are a way to improve the readability of a program. Comments can be used to include additional information about a program like the author of the code, hints about a function/construct, etc. Comments are ignored by the compiler.

JavaScript supports the following types of comments −

- **Single-line comments (//)** − Any text between a // and the end of a line is treated as a comment.
- **Multi-line comments (/* */)** − These comments may span multiple lines.

Example

```
//this is single line comment
/* This is a
Multi-line comment
*/
```

Your First JavaScript Code

Let us start with the traditional “Hello World” example".

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

The program can be analyzed as −

- Line 1 declares a variable by the name message. Variables are a mechanism to store values in a program.
- Line 2 prints the variable’s value to the prompt. Here, the console refers to the terminal window. The function log () is used to display the text on the screen.

Executing the Code

We shall use Node.js to execute our code.

- **Step 1** − Save the file as Test.js
- **Step 2** − Right-click the Test.js file under the working files option in the project-explorer window of the Visual Studio Code.
- **Step 3** − Select Open in Command Prompt option.
- **Step 4** − Type the following command in Node's terminal window.

```
node Test.js
```

The following output is displayed on successful execution of the file.

```
Hello World
```

Node.js and JS/ES6

ECMAScript 2015(ES6) features are classified into three groups −

- **For Shipping** − These are features that V8 considers stable.
- **Staged Features** − These are almost completed features but not considered stable by the V8 team.
- **In Progress** − These features should be used only for testing purposes.

The first category of features is fully supported and turned on by default by node. Staged features require a runtime - - harmony flag to execute.

A list of component specific CLI flags for Node.js can be found here − https://nodejs.org/api/cli.html

The Strict Mode

The fifth edition of the ECMAScript specification introduced the Strict Mode. The Strict Mode imposes a layer of constraint on JavaScript. It makes several changes to normal JavaScript semantics.

The code can be transitioned to work in the Strict Mode by including the following −

```
// Whole-script strict mode syntax
"use strict";
 v = "Hi!  I'm a strict mode script!";  // ERROR: Variable v is not declared
```

In the above snippet, the entire code runs as a constrained variant of JavaScript.

JavaScript also allows to restrict, the Strict Mode within a block's scope as that of a function. This is illustrated as follows −

```
v = 15
function f1() {
   "use strict";
   var v = "Hi!  I'm a strict mode script!";
}
```

In, the above snippet, any code outside the function will run in the non-strict mode. All statements within the function will be executed in the Strict Mode.

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.

Variable Hoisting and Function Hoisting are explained in the subsequent chapters.