ECMASCRIPT6(ES6+) SYNTAX

{ES6}

ECMASCRIPT 6

A guide by ***Prophete ISINGIZWE***

ECMAScript: is a standard for scripting languages, including JavaScript, JScript, and ActionScript. It is also best known as a JavaScript standard intended to ensure the interoperability of web pages across different web browsers.

ES6, also known as ECMAScript 2015, is a significant update to JavaScript that was released in 2015. It includes several new features, such as let and const declarations, arrow functions, template literals, classes and much more we are gonna discuss later, that make JavaScript code more concise, expressive, and maintainable.

1. Intro

What is ECMA & What versions of EcmaScript do we have?.

Definition

ECMA - European Computer Manufacturers Association (Neutral party for setting standards in the IT industry since 1961)

➔ First ECMAScript (June 1997)
➔ Second ECMAScript (June 1998)
➔ Third ECMAScript (June 2002)
➔ Fourth ECMAScript (Never published)
➔ Fifth ECMAScript (December 2009)
➔ Fifth.1 ECMAScript (June 2011)
➔ Sixth ECMAScript (June 2015)

Facts about javascript

- Javascript was written in 10 days.
- High Adoption Rate: JavaScript is one of the most widely used programming languages on the web. It is supported by all major web browsers, making it accessible to billions of users globally. Its high adoption rate is due to its ability to add interactivity to websites and web applications.
- Client-Side and Server-Side Language: JavaScript is unique in that it can be used both as a client-side language (running in the user's browser) and as a server-side language (running on the server). On the client-side, JavaScript is responsible for handling user interactions and manipulating the web page's content. On the server-side, it can be used with platforms like Node.js to build scalable and efficient server applications.
- Asynchronous Programming: JavaScript is known for its support of asynchronous programming through mechanisms like callbacks, promises, and async/await. This is crucial for handling non-blocking operations like fetching data from APIs, performing file I/O, or handling user interactions without freezing the application.
- ECMAScript Standard: JavaScript is an implementation of the ECMAScript standard, which is maintained by Ecma International. The ECMAScript standard defines the language's specifications and features, ensuring consistency across different implementations. Different versions of JavaScript, such as ES6 (ES2015), ES7 (ES2016), and so on, are aligned with different ECMAScript editions.

What we'll be covering.

- ➔ Template literals
- ➔ Variables (let/const)
- ➔ Objects & Object Literals
- ➔ Array Methods
- ➔ Arrow functions
- ➔ Destructuring
- ➔ Spread operators
- ➔ Rest operators
- ➔ Default params
- ➔ Loops
- ➔ Sets
- ➔ this
- ➔ Classes
- ➔ Modules
- ➔ Async & Await

Template literals

*** The 3 types of ways to construct a sentence(string) in js is by the use of: ‘’, “” and ``. Which is referred to as string literals.*

Variables defined with “var” keyword can still be accessed outside their declaration block within the same function.

While variables defined with the “let” and “const” keyword (ES6+ way of variable declaration) are only accessible within their declaration block.

The 3 types of string literals: 👇

‘’	“”	``
Can be combined with other string literal types	Can be combined with other string literal types	Can be combined with other string literal types
Used from past EcmaScript versions until now	Used from past EcmaScript versions until now	Introduced as an ES6+ syntax for string declarations. (string concatenation)

Variables (let/const)

*** The main difference between “var”, “let” and “const” is scoping/block accessibility.*

Variables defined with “var” keyword can still be accessed outside their declaration block within the same function.

While variables defined with the “let” and “const” keyword (ES6+ way of variable declaration) are only accessible within their declaration block.

Variables declared with: 👇

var(👎)	***let(👍)***	***const(👌)***
Accessed outside their declaration block	Only available inside their declaration block	Only available inside their declaration block
Can be re-assigned	Can be re-assigned	Can not be re-assigned
ES5 (and before) syntax for declaring variables	ES6+ syntax for declaring variables that can be re-assigned	ES6+ syntax for declaring variables that are static. (constants)

Objects

*** In JS, objects are a collection of key/value pairs.*

In Object Oriented Programming (OOP), a function declared inside an object is referred to as a "Method"

With ES5 and less, methods are annotated with their name, a colon (:) and the "function" that declares what the method does.

With ES6+ method annotation were simplified, by the using the name along with brackets and curly brackets.

Method declaration: 👇

ES5(👎)	***ES6+(👍)***

const objectName = {
  Name: function(){}
}
	const objectName = {
	  Name(){}
	}

Objects Literals

*** Prior to ES6, an object literal is a collection of name-value pairs .*

ES6 allows you to eliminate the duplication when a property of an object is the same as the local variable name by including the name without a colon and value.

Internally, when a property of an object literal only has a name, the JavaScript engine searches for a variable with the same name in the surrounding scope. If the JavaScript engine can find one, it assigns the property the value of the variable.

Method declaration: 👇

ES5(👎)	***ES6+(👍)***
function objectName(param) { param: param }	const objectName(param) { param }

Arrays in ES6+

*** in programming, an array is a collection of items, or data, stored in contiguous memory locations*

+ Many methods were introduced in ES6+ to improve the manipulation of arrays in javascript

!! In Object Oriented Programming(OOP) terms, a function declared inside an object is referred to as a "Method".

Mostly used array methods: 👇

- ➢ .find()
- ➢ .includes()
- ➢ .filter()
- ➢ .map()
- ➢ .reduce()
- ➢ .some()
- ➢ .flat()
- ➢ .forEach()
- ➢ .of()
- ➢ .every()

Array Methods in ES6+

** ES6+ provides a new kind of way to check if an element is present in an array.

Array.prototype.find(): This method returns the first element in the array that satisfies the provided testing function. If no element is found, it returns undefined.

Array.prototype.includes(): This method checks whether an array includes a specific element, and it returns a boolean value indicating the result.

Array.prototype.filter(): It creates a new array with all elements that pass the test implemented by the provided function.

Array.prototype.map(): This method creates a new array by applying a function to each element in the original array.

Array.prototype.reduce(): This method applies a reducer function to each element of the array, resulting in a single value.

Array.prototype.some(): It checks if at least one element in the array passes the test implemented by the provided function. It returns a boolean value.

Array.prototype.flat(): It creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

Array.prototype.forEach(): It executes a provided function once for each array element.

Array.of(): This method creates a new array with the given arguments as elements.

Array Methods in ES6+

Array.prototype.flatMap(): This method first maps each element using a mapping function and then flattens the result into a new array.

Array.prototype.keys(): This method returns a new array iterator that contains the keys (indices) of each element in the array.

Array.prototype.every(): This method checks if all elements in the array pass the test implemented by the provided function. It returns a boolean value.

Array.from(): It creates a new array from an array-like or iterable object.

Array.prototype.values(): It returns a new array iterator that contains the values of each element in the array.

Array.prototype.entries(): This method returns a new array iterator that contains the key-value pairs for each element in the array.

What to know(👌)

These array methods have proven to be very useful in JavaScript development and have improved the readability and maintainability of code when working with arrays.

Arrow functions

*** ES6 arrow functions provide you with an alternative way to write a shorter syntax compared to the function expression.*

++ With array functions, you can transform a 3 - 5 lines function into one single line that does the exact same thing.

!! Arrow functions helps in noise reduction when writing code.

With arrow functions: 👇

***Syntax:* name = () => {} (👌)**
When a function has only 1 parameter, the brackets can be omitted.
When a function has more than 1 or no parameters, the brackets should always be used.
When a function has only 1 return, the return keyword can be omitted.

Object destructuring

👉*Object destructuring is a feature introduced in ES6 (ECMAScript 2015) and is a convenient way to extract properties from objects and assign them to variables. It allows you to unpack values from objects and create references to those values using a concise and expressive syntax.*

The syntax for object destructuring is as follows:

```
const { property1, property2, ... } = object;
```

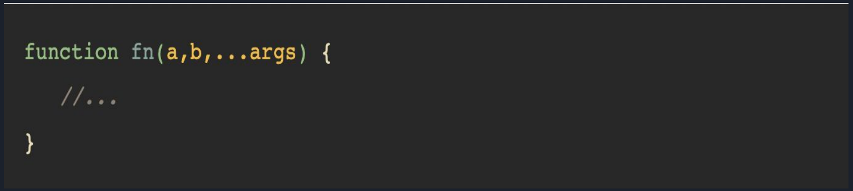
Here's a brief explanation of each part:

- const: This keyword (or let/var) is used to declare variables. The variables created through destructuring will be constants (immutable) if declared with const.
- {}: The curly braces are used to denote the object destructuring pattern.
- property1, property2, ...: These are the variable names that will be used to reference the corresponding properties of the object.
- object: The source object from which properties will be extracted.

Rest operators

** ES6+ provides a new kind of parameter so-called “rest parameter” that has a prefix of 3 dots (...).

+ *A rest parameter allows you to represent an indefinite number of arguments as an array.*



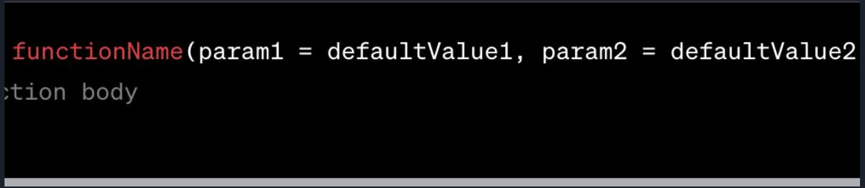
```
function fn(a,b,...args) {
   //...
}
```

What to know(👌)
The “...args” returns an array
If only 2 params are provided(in case of the function in the picture), the rest parameter will return an empty array
The rest parameters must appear at the end of the argument list. If passed in between other named params, it will result in an error:

Default Parameters

👉🏾 *Default parameters, also known as default arguments or default values, are a feature introduced in ES6 (ECMAScript 2015) that allows you to assign default values to function parameters. When a function is called and a parameter is not provided or is explicitly set to undefined, the default value specified in the function declaration is used instead.*

+ Default parameters are useful for handling cases where you want to ensure that a parameter has a value even if the caller doesn't provide it explicitly. They help make functions more robust and prevent errors related to missing or undefined arguments.



```
functionName(param1 = defaultValue1, param2 = defaultValue2
:tion body
```

In the syntax of the image aside:

functionName: The name of the function.

param1, param2, ...: The names of the function parameters that you want to assign default values to.

defaultValue1, defaultValue2, ...: The default values that will be used for the corresponding parameters if not provided by the caller.

Loops

👉 *ES6+ introduced two new looping constructs that provide more concise and expressive ways to iterate over arrays and objects: the “for...of” loop and the “for...in” loop.*

for...of loop:

The for...of loop is used to iterate over elements of an iterable object, such as arrays, strings, maps, sets, etc. It provides a simpler and more readable alternative to traditional for loops when you need to iterate over the values of a collection.

```
for (const element of iterable) {
  // Code to be executed for each element
}
```

for...in loop:

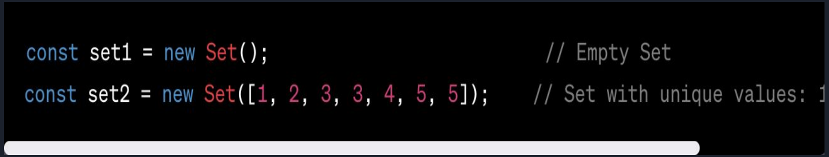
The for...in loop is used to iterate over the enumerable properties of an object. It provides a way to access the keys or property names of an object, which can be useful when you need to perform specific operations based on object properties.

```
for (const key in object) {
  // Code to be executed for each property
}
```

Sets()

👉In ES6 (ECMAScript 2015) and beyond, JavaScript introduced a new built-in data structure called "Set." A Set is a collection of unique values, and it can hold various types of data, such as primitive values or object references. Sets are especially useful when you need to ensure that no duplicates exist within the collection.

Here's how you create and work with Sets in JavaScript:



```
const set1 = new Set();                          // Empty Set
const set2 = new Set([1, 2, 3, 3, 4, 5, 5]);    // Set with unique values: 1
```

What to know(👌)

You can add elements in a set:
`set1.add('a')`

You can delete elements in a set:
`set1.delete('a')`

You can check the size of a set:
`set1.size`

You can check if a set has particular elements:
`set1.has('a')`

You can check loop a set:

```
for (const item of mySet)
{ console.log(item); }
```

You can convert a Set into an array using the Array.from() method or using the spread operator ....

```
const mySet = new Set([1, 2, 3, 4, 5]);
const myArray = Array.from(mySet);
const myArray = [...mySet];
```

"this" keyword

👉*In ES6 (ECMAScript 2015) and beyond, the `this` keyword behaves slightly differently compared to earlier versions of JavaScript, especially in arrow functions. Understanding the context of this is crucial because it determines what object the function refers to.*

- ❏ Regular Functions: In regular functions, the value of this depends on how the function is called. It is dynamically determined by the context in which the function is invoked.
- ❏ Arrow Functions: Arrow functions have a different behavior regarding the this keyword. Unlike regular functions, arrow functions do not have their own this context. Instead, they capture the this value of the surrounding lexical scope at the time of their creation. This makes arrow functions very useful for callbacks and situations where you want to preserve the value of this.

What to know(👌)

Global Context: When a function is called in the global scope, `this` refers to the global object (in a browser environment, it refers to the `window` object).

Object Method: When a function is called as a method of an object, `this` refers to the object on which the method is called.

Explicit Binding: You can use methods like `call()`, `apply()`, or `bind()` to explicitly set the value of `this` within a function.

Classes

👉*In ES6 (ECMAScript 2015) and later versions, JavaScript introduced a new way to create and work with objects through the use of class syntax. Classes in JavaScript are a syntactical sugar over the prototype-based inheritance model, providing a more familiar and organized approach to object-oriented programming.*

The class syntax allows developers to define a blueprint for creating objects with shared properties and methods. These objects are referred to as instances of the class. It simplifies the process of creating objects and managing their behavior.

Classes are made up of usually 4 parts and here's a brief explanation of each part:

★ class: The class keyword is used to declare a new class.

★ ClassName: Replace ClassName with the name of the class you want to define. The naming convention for classes follows the same rules as for variable names (e.g., start with a letter, no spaces or special characters except underscores).

★ constructor(): This is a special method called when an instance of the class is created. It is used to set up the initial state of the object and accepts any parameters you want to pass during object creation. The this keyword is used to refer to the current instance of the class.

★ Methods: Inside the class body, you can define various methods that represent the behavior of the objects created from the class. These methods are similar to regular functions, but they are associated with the class and its instances.

Modules

👉*Modules in JavaScript are a way to organize and structure code by dividing it into smaller, reusable, and independent pieces. Prior to ES6 (ECMAScript 2015), JavaScript did not have native support for modules, and developers relied on various patterns and techniques to achieve modularity.*
The syntax for object destructuring is as follows:

With ES6 and later versions, JavaScript introduced native support for modules through the use of the import and export keywords. This standardized module system provides a more organized and maintainable approach to managing dependencies and code sharing in larger projects.

Exporting from a Module:

➢ *To make variables, functions, or classes available to other parts of the application, you can use the `export` keyword*

Importing from a Module:

➢ To use the exported values from another module, you can use the import keyword followed by the desired variable, function, or class name and the path to the module.

Async & Await

👉Async/await is a powerful feature introduced in ES2017 (ES8) that simplifies asynchronous code in JavaScript. It allows you to write asynchronous operations in a synchronous style, making the code more readable and easier to understand. Async/await is built on top of Promises and provides a more intuitive way to handle asynchronous tasks.

async Function:

To define an asynchronous function, you use the async keyword before the function declaration. The async keyword tells the JavaScript engine that the function contains asynchronous operations, and it will always return a Promise.

await Keyword:

The await keyword is used within an async function to wait for the resolution of a Promise. It pauses the execution of the function until the Promise is resolved or rejected. If the Promise is resolved, the value of the resolved Promise is returned. If the Promise is rejected, an exception is thrown.

What to know(👌)

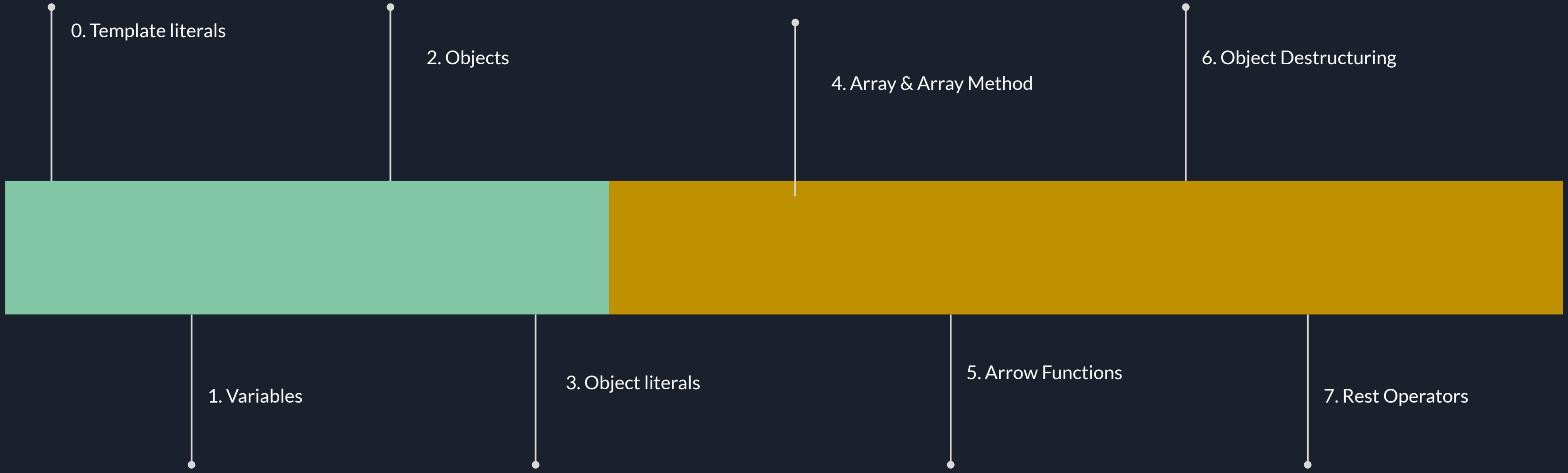
- **Error Handling**: Async/await allows you to use traditional try...catch blocks for error handling, making it easier to handle asynchronous errors in a synchronous way.

- **Use with Promises**: Async/await can be used with existing Promise-based code. You can await a Promise within an async function, and you can also return a Promise from an async function.

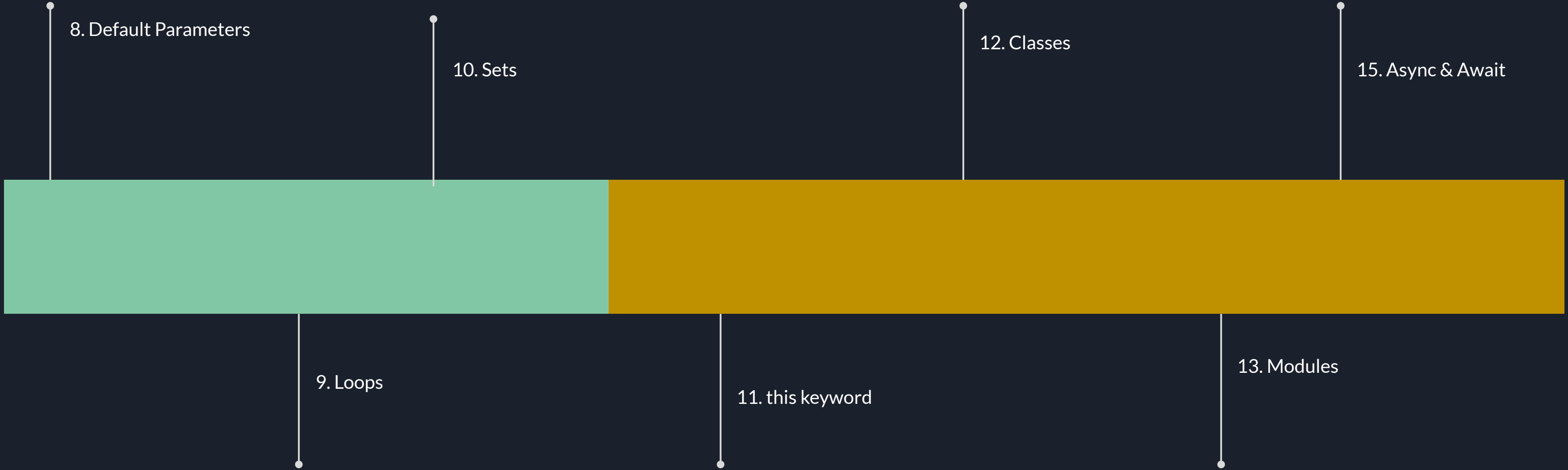
- **Sequential and Parallel Execution**: With async/await, you can write asynchronous code that executes sequentially or in parallel, depending on your needs.

Async/await provides a cleaner and more concise way to work with asynchronous code in JavaScript. It has become a popular choice for managing asynchronous tasks, especially when dealing with APIs, database operations, and other asynchronous operations.

Milestones


0. Template literals
1. Variables
2. Objects
3. Object literals
4. Array & Array Method
5. Arrow Functions
6. Object Destructuring
7. Rest Operators

Milestones


8. Default Parameters
9. Loops
10. Sets
11. this keyword
12. Classes
13. Modules
15. Async & Await

4. ***Closing***

- ➔ Q&As
- ➔ Additions
- ➔ Suggestions

What we saw!

History of EcmaScript

Facts about Javascript

ES6+ syntax & usage