What is JS?

3 core languages used to build a web app - HTML, CSS & JavaScript.

1. **HTML** - HTML is the standard markup language for creating Web pages. All those buttons that you see on a website!
2. **CSS** - CSS is the language we use to style a Web page. All those colors, animations that you see on a like button!
3. **JS** - JavaScript is a scripting language of the web that allows you to do Interactivity with User-Events, implement Conditions and Validations, Dynamic updates in a Web Page, etc. All the logic when you hit that like button!

Syntax

Checkout the below code for example. It contains all the above mentioned languages.

You can see various html tags like - `head`, `title`, `body`, etc.
You can see css styles enclosed in `<style>` html tags.
You can see js code enclosed in various `<script>` html tags.

```
<html lang="en">
  <head>
    <title>Introduction to JS</title>

    <style>
      body {
        font-family: arial;
      }
    </style>

    <script>
      // Write all JavaScript code here
      alert("welcome to JavaScript");
    </script>
  </head>

  <body>
    Content

    <script>
      /* Write all JavaScript code here */
      console.log("welcome to JavaScript");
    </script>
  </body>
</html>
```

Let's learn how to write JavaScript now!

The syntax of JavaScript is the set of rules that define a correctly structured JavaScript program. Let's study some of the building blocks of JavaScript code:

Comments

1. The comments are a meaningful way to deliver the message (description to code) for others/for future reference and to understand/follow the code statements/lines
2. Comments are special lines written for other developers as a reference purpose and browser ignores while interpreting

There are 2 types of comments -

1. Singleline comment - starts with `//`
2. Multiline comment - starts with `/**` and ends with `**/`

```
// This is a single line comment.
// This is another single line comment.

/**
  This is a multiline comment
  Everything enclosed inside this will be a comment.
**/
```

Statement

1. One line of JavaScript Code is one JavaScript Statement / Instruction / Command
2. A Statement is a piece of code that expresses an action to take place
3. Statements are separated by `Semicolon;` so we may write many statements in a line

```
/**
  These are just comments
  This code contains 2 statements.
  The first statement assings "Javascript" value to a variable named
courseName
  The second statement logs the courseName in the console.
**/
let courseName = "Javascript";
console.log(courseName);
```

JS Code blocks

1. JavaScript commands/statements/code can be grouped together in code blocks, inside curly brackets `{...}`
2. Grouped statements/lines form code blocks
3. The purpose of code blocks is to define statements to be executed together like a single JavaScript command

4. An often occurrence of a code block in JavaScript is a JavaScript `function`

```
/**
  Below is a code block wrapped inside a function named - sayHello
  When you call a function, it executes all the statements in that block
**/
function sayHello() {
  alert("Hello All! Welcome to JavaScript!!");
}

/**
  Below is an other example of a function named - showTotal
**/
function showTotal() {
  var num1 = 10;
  var num2 = 20;
  var total = num1 + num2;
  alert("Total is : " + total);
}

/**
calling above defined functions, to execute all the statements in a block
at once.
**/
sayHello();
showTotal();
```

Keywords are reserved words that are part of the syntax in the programming language. For example,

Here is the list of some keywords available in JavaScript.

const	let	var	for	function
return	if	else	try	catch
do	while	break	continue	await
async	switch	class	import	new

These words cannot be used as identifiers. These keywords are used for different purposes to complete your code.

```
const a = "hello";
```

Here, `const` is a keyword that denotes that **a** is a constant.
`a` is an identifier and it cannot be from the list of keywords present in js.

We will learn about the meaning of individual keywords as we progress in the course.

An identifier is a name that is given to entities like variables, functions, class, etc.

1. Identifier names must start with either a **letter**, an underscore **_**, or the dollar sign **$**. For example,

```
//valid
const a = "hello";
const _a = "hello";
const $a = "hello";
```

2. Identifier names cannot start with numbers. For example,

```
//invalid
const 1a = 'hello'; // this gives an error
```

3. JavaScript is **case-sensitive**. So y and Y are different identifiers. For example,

```
const y = "hi";
const Y = 5;
console.log(y); // hi
console.log(Y); // 5
```

4. Keywords cannot be used as identifier names. For example,

```
//invalid
const new = 5; // Error! new is a keyword.
```

> Note: Though you can name identifiers in any way you want, it's a good practice to give a descriptive identifier name.
> If you are using an identifier for a variable to store the number of students, it is better to use **students** or **numberOfStudents** rather than x or n.

All modern browsers have a web console for debugging. The `console.log()` method is used to write messages to these consoles. For example, the following code will write a message to the console:

```
// assigning 44 to a variable named sum
let sum = 44;
// logging the value of sum in console
console.log(sum); // 44
```

When you run the above code, 44 is printed on the console.

Syntax

```
console.log(message);
```

Here, the message(can be a string indicating message wrapped inside "" , '' or ``) refers to either a variable or a value(number, string, etc.).

Example

Using the "", '', ``:

```
console.log("Hello World");
console.log('Hello World');
console.log(`Hello World`);
```

In order to insert a variable into a string, you can use the `${}` syntax called template literals. For example:

Syntax : `console.log(`example-string ${variable-name}`);`

```
let name = "John";
console.log(`Hello ${name}`);
```

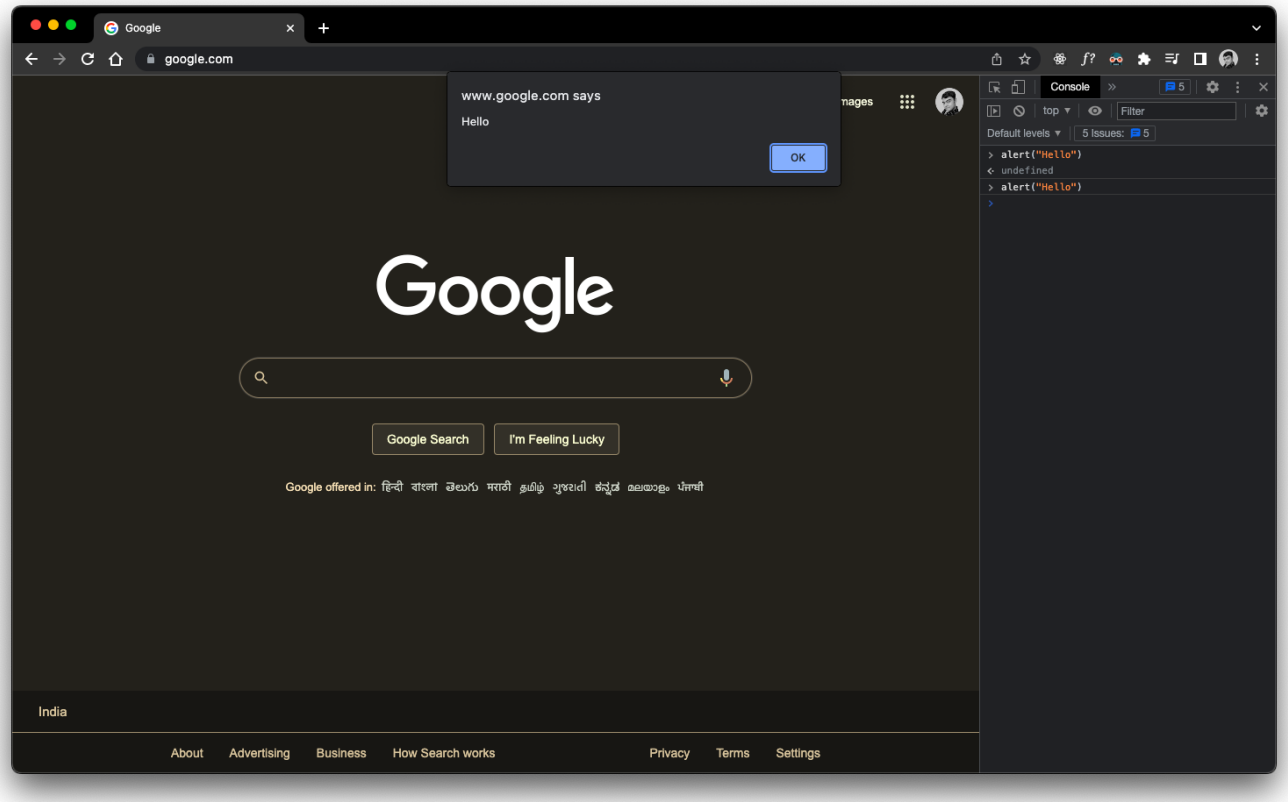
Simmilarly, you can use the `${}` syntax to insert a value into a string:

```
let value = 44;
console.log(`The value is ${value}`);
```

Alert function in JavaScript

```
let a = "Hello World";
alert(a);
```

The alert function is used to display a message in an alert box in a browser.



Example

```
let a = "This is an alert";
alert(a);
```

JavaScript

JavaScript is one of the most popular programming languages in the world.
It helps to build interactive web pages, games, and applications.
With the introduction of ECMAScript 2015, JavaScript has really become a powerful language.
And now with the help of Node.js, it is possible to build web applications with JavaScript both client-side and server-side.

Variables in JavaScript

JavaScript is a dynamically typed language, and is loosely typed which means there little to no need to define scrict types to its variables.

The `var` keyword

In Js, there are numerous ways to define variables, one of which is `var`.
The `var` keyword is used to define a variable.
An uninitialized var variable is undefined.

Example

```
var x = 10;
console.log(x);
```

Output:
10

```
var a = "Hello World";
```

The above code will create a variable called `a` and assign the value `Hello World` to it.

Variable Scope

Variables defined with the `var` keyword are created in the global scope.

```
var a = "Hello World";
var b = "Hello World";
```

The above code will create two variables called `a` and `b` and assign the value `Hello World` to them.

```
console.log(a);
console.log(b);
```

The above code will print the value of `a` and `b` in the console.

Now the thing with var is that it creates a variable in the global scope.

Quick Quiz

```

```

```
var a;
```

- What is the value of `a`?

  > `undefined`

The `let` keyword

The `let` keyword is used to define a variable. The difference between `var` and `let` is that `let` creates a variable in the scope of the block it is defined in.

A variable defined using the `let` keyword is created in the block it is defined in and is not available outside of that block.

Example

```
let x = 10;
console.log(x);

function myFunction() {
  console.log(x);
  let x = 20;
  console.log(x);
}

myFunction();
```

Output:
10
Uncaught ReferenceError: Cannot access 'x' before initialization
at myFunction (REPL8:2:15)
20

- Note that the above code will throw an error because `x` is not defined in the global scope. It is only defined in the `myFunction` function.
- Functions are not hoisted, so `myFunction` will not be defined until the end of the code.
- Functions as an concept will be discussed later, for now consider them as a block of code that can be executed on being called.

The `const` keyword

The `const` keyword is used to define a constant variable. A constant variable is a variable that cannot be reassigned.
A constant variable is created in the global scope, and is available everywhere.

Example

```
const x = 10;
console.log(x);
function myFunction() {
  console.log(x);
  const x = 20;
  console.log(x);
}
myFunction();
```

Output:
10
10
20

- Here `x` is defined as a constant variable in the global scope, hence it can be accessed anywhere even in the `myFunction` function.

The `let` keyword vs `const` keyword

The `let` keyword is used to define a variable. The difference between `var` and `let` is that `let` creates a variable in the scope of the block it is defined in, and `const` creates a constant variable in the global scope.

Comments in JavaScript

Comments in JavaScript are used to explain the code, and are used to make the code easier to read.

- Single line comments start with `//`
- Multi-line comments start with `/*` and end with `*/`
- Comments can be nested
- Comments can be used to disable code

```
// Single line comment
/*
Multi-line comment
*/
```

Quick Quiz

```
// var a;
```

Output of the above code?

> nothing is printed to the console

Data Types in JavaScript

- Data types are used to define the type of data that a variable can hold
- In JavaScript, there are six data types:
    - Number
    - String
    - Boolean
    - Object
    - Array
    - Function
- These data types are classified as primitive data types (or simple data types) and complex data types (which are objects).
- Primitive data types are the simplest data types.
- Number : A number is a data type that represents a numeric value.

> A number can be a whole number (integer), a decimal number (float), or a negative number.

- String : A string is a data type that represents text.

> A string can be a single character, a word, or a sentence.

- Boolean : A boolean is a data type that represents a true or false value.

> A boolean can be true or false.

- Null : A null is a data type that represents the intentional absence of a value.

> A null value is often used to represent the absence of a value and can be explicitly assigned to a variable.

- Undefined : A undefined is a data type that represents the intentional absence of a value.

> A undefined value is often used to represent the absence of a value which is not yet defined.

- Object : An object is a data type that represents a collection of data.

> An object is a collection of data that is stored in key-value pairs.

- Array : An array is a data type that represents a list of data.

> An array is a list of data that is stored in a sequential order.

- Function : A function is a data type that represents a block of code.

> A function is a block of code that is used to perform a specific task.

Quiz

```
var number = 10;
var string = "Hello";
var boolean = true;
var nullValue = null;
var undefinedValue = undefined;
var object = {};
var array = [];
var functionValue = function () {};
```

1. What is the data type of the variable `number`?
2. What is the data type of the variable `string`?
3. What is the data type of the variable `boolean`?
4. What is the data type of the variable `nullValue`?
5. What is the data type of the variable `undefinedValue`?
6. What is the data type of the variable `object`?
7. What is the data type of the variable `array`?
8. What is the data type of the variable `functionValue`?

Answer

1. number
2. string
3. boolean
4. null
5. undefined
6. object
7. array
8. function

Some extra notes

- Strings enclosed in double quotes if include a double quote, it will be treated as the end of the string, hence escape the double quote using the escape character "

Arithmatic Operators

- As the name suggests, the arithmatic operators are used to perform arithmatic operations.
- The arithmatic operators are:
    - Addition (+)
    - Subtraction (-)
    - Multiplication (*)
    - Division (/)
    - Modulus (%)
    - Exponentiation (**)
    - Increment (++)
    - Decrement (--)
- The addition operator is used to add two numbers, just like

```
var a = 10;
var b = 20;
var sum = a + b;
document.write(sum);
```

> document.write() is a function that is used to write text to the browser, in this case, it will write the sum of a and b to the browser.

- The subtraction operator is used to subtract two numbers, just like

```
var a = 10;
var b = 20;
var difference = a - b;
document.write(difference);
```

- The multiplication operator is used to multiply two numbers, just like

```
var a = 3;
var b = 5;
var multiply = a * b;
document.write(multiply);
```

- The division operator is used to divide two numbers, just like

```
var a = 10;
var b = 20;
```

```
var divide = a / b;
document.write(divide);
```

- The modulus operator is used to find the remainder of two numbers, just like

```
var a = 3;
var b = 5;
var modulus = a % b; // 3
document.write(modulus);
```

> The remainder of two numbers (a % b) is the number that remains after the division of a by b.

- The exponentiation operator is used to raise a number to a power, just like

```
var a = 3;
var b = 5;
var exponent = a ** b; // 243
document.write(exponent);
```

- The increment operator is used to increment a number by 1, just like

```
var a = 10;
a++;
document.write(a); // 11
```

- The decrement operator is used to decrement a number by 1, just like

```
var a = 10;
a--;
document.write(a); // 9
```

Quiz Time!

- What is the value of answer after executing the following code?

```
var a = 10;
var b = 20;
var answer = a + (10 * b) / 2;
```

- Answer : `answer = a + (10 * b) / 2 = 10 + 200 / 2 = 10 + 100 = 110`

Assignment Operators

- We use assignment operators to assign values to variables in JavaScript.
- There are many assignment operators in JavaScript, which are:

Operator	Example	Is equivalent to
=	a = 10;	a = 10;
+=	a += 10;	a = a + 10;
-=	a -= 10;	a = a - 10;
*=	a *= 10;	a = a * 10;
/=	a /= 10;	a = a / 10;
%=	a %= 10;	a = a % 10;
**=	a **= 10;	a = a ** 10;

> Note : you can use multiple assignment operators in a single statement `a += b -= c %= 10;`

Quiz time!

- What is the value of answer after executing the following code?

```
var number = 2;
number *= 5;
document.write(number);
```

> Answer : `number *= 5 = 2 * 5 = 10`

Comparison Opertors

- We can use comparison operators to compare values and expressions.
- We get either true or false.
- For example (==) checks whether the operands' values are equal.

```
var a = 10;
console.log(a == 10); // true
```

> In javascript we have two types of equality operators (== is loose and === is strict).
> The loose equality operator (==) is used to compare values.
> The strict equality operator (===) is used to check whether the operands' values and types are equal.

```
var a = 10;
var b = "10";
console.log(a == b); // true
console.log(a === b); // false
```

- Here for the strict equality operator (===) we get false because the types are not equal (number and string).

List of Comparison Operators

Operator	Description	Example
==	Equal	a == b
===	Strict Equal	a === b
!=	Not Equal	a != b
!==	Strict Not Equal	a !== b
>	Greater Than	a > b
<	Less Than	a < b
>=	Greater Than or Equal	a >= b
<=	Less Than or Equal	a <= b

Quiz Time!

```
var a = 1024;
var b = "1 GB";
```

```
console.log(a == b); // ?
```

> What is the result of the above code? : false

Logical or Boolean Operators

- The logical or boolean operators are used to combine conditions or expressions and return a single boolean value
- The logical operators are (AND, OR, NOT)

Logical Operator	Description	Example
&&	Logical AND (returns true if both operands are true)	a && b
||	Logical OR (returns true if either of the operand is true)	a | b
!	Logical NOT (returns true if the oparand is false and vice versa)	!a

- These operators are use to perform boolean operations on the operands, and return a single boolean value

Quiz time!

```
var a = true;
var b = false;
var c = true;
var o1 = (a && !b) || !c;
```

> Output: true

String Operations

- The basic operations on a string are:
    - Concatenation
    - Comparison
    - Searching
    - Replacing
    - Splitting
    - Trimming
    - Case conversion
- The concatenation operation is used to combine two or more strings together
- The comparison operation is used to compare two strings
- The searching operation is used to find a string within another string
- The replacing operation is used to replace a string with another string
- The splitting operation is used to split a string into an array of substrings
- The trimming operation is used to remove whitespace from the beginning and end of a string
- The case conversion operation is used to convert a string to uppercase or lowercase

Example

```
var str1 = "Hello";
var str2 = "World";
var str3 = str1 + " " + str2; // Concatenation
var str4 = str3.toUpperCase(); // Case conversion
var str5 = str3.toLowerCase(); // Case conversion
var str6 = str3.replace("World", "Universe"); // Replacing
var str7 = str3.split(" "); // Splitting
var str8 = str3.trim(); // Trimming
```

Quiz time!

```
var str1 = " Hello ";
var str2 = " World ";
var str3 = str1.trim() + " " + str2.trim();
var str4 = str3.toUpperCase();
```

> Output: HELLO WORLD

The if statement

- The if statement is used to execute a block of code if a condition is true

```
if (condition) {
  // code to execute
}
```

Example

```
var a = 100;
var b = 20;
if (a > b) {
  // Condition is true so the if block gets executed
  console.log("a is greater than b");
}
```

The else statement

- When the condition is false for the existing if clause, the else block gets executed

```
if (condition) {
  // code to execute
} else {
  // code to execute if condition is false
}
```

Example

```
var age = 12;
if (age > 18) {
  console.log("You are an adult");
} else {
  // Condition is false so the else block gets executed
  console.log("You are not an adult");
}
```

The else if statements

- The else if statement is a special case of the if statement, when the if statement is not true it runs the else if statement.

```
if (condition) {
  // code to run if condition is true
} else if (condition) {
  // code to run if condition is true
} else {
  // code to run if condition is false
}
```

Example

```
var a = 1;
var b = 2;
if (a > b) {
  console.log("a is greater than b");
} else if (a < b) {
  console.log("a is less than b");
} else {
  console.log("a is equal to b");
}
```

The switch statement

- The switch statement is a more powerful way to write a conditional statement, it takes a value and then checks if it matches with a case provided.
- The switch statement is a bit like a switch in other languages.
- The switch statement syntax is:

```
switch (value) {
  case value1:
    // code block
    break;
  case value2:
    // code block
    break;
  default:
  // code block
}
```

- The default case is optional, but it is recommended to use it.
- It helps to give a case to the user when none of the cases get matched.

Example

```
var day = "Tuesday";
switch (day) {
  case "Monday":
    console.log("Today is Monday");
    break;
  case "Tuesday":
    console.log("Today is Tuesday");
    break;
  case "Wednesday":
    console.log("Today is Wednesday");
    break;
  case "Thursday":
    console.log("Today is Thursday");
    break;
  case "Friday":
    console.log("Today is Friday");
    break;
  case "Saturday":
    console.log("Today is Saturday");
    break;
  case "Sunday":
    console.log("Today is Sunday");
    break;
  default:
```

```
    console.log("Today is not a day of the week");
}
```

- Here the control variable is `day`, and the value of the variable is `"Tuesday"`, so the switch statement will check if the value of the variable matches with any of the cases and it matches with the case `"Tuesday"`, so it will execute the code block and print `"Today is Tuesday"`.

The for loop

- The for loop comes in a class of iterative statements.
- It is used to iterate over a block of code a specified number of times.

Syntax

```
for (initialization; condition; final - expression) {
  // code block
}
```

- Here, the initialization is the code that is executed before the loop starts.
- The condition is the code that is executed to check if the loop should continue.
- The final expression is the code that is executed after the loop finishes, usually used to update the control variable.

Example

- Logging out the numbers from 1 to 10

```
var i = 0;
for (i = 0; i < 10; i++) {
  console.log(i);
}
```

- Here the control variable is `i`, and the value of the variable is `0`, so the loop will start at `0` and it will check if the value of the variable is less than `10`, so it will execute the code block and print `0`, then it will update the value of the variable to `1` and it will check if the value of the variable is less than `10`, so it will execute the code block and print `1`, then it will update the value of the variable to `2` and it will check if the value of the variable is less than `10`, so it will execute the code block and print `2`, and so on.

The while loops

- The while loop is a type of loop that executes a block of code as long as a condition is true.
- Its used when we do not know how many times the loop will execute, it is an entry controlled loop.

Syntax

```
while (condition) {
  // code block
}
```

Example

- Logging out the numbers from 1 to 10

```
var i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
```

The do while loop

- The do while loop is a type of loop that executes a block of code at least once, and then checks the condition.
- It is used when we know how many times the loop will execute, it is an exit controlled loop.

Syntax

```
do {
  // code block
} while (condition);
```

Example

- Logging out the numbers from 1 to 10

```
var i = 0;
do {
  console.log(i);
  i++;
} while (i < 10);
```

The break and continue statements

- The break statement is used to exit a loop.
- The continue statement is used to skip the rest of the code in a loop.

Syntax

```
var i = 0;
for (; i < 10; i++) {
  if (i === 5) {
    break;
  }
  console.log(i);
}
```

- The break statement here is used to exit the loop when i is equal to 5, so the code ony runs till i is equal to 5.

```
var i = 0;
for (; i < 10; i++) {
  if (i === 5) {
    continue;
  }
  console.log(i);
}
```

- Here the continue statement is used to skip the rest of the code in the loop when i is equal to 5.

Functions in JavaScript

A function is a block of code that performs a specific task.

- Functions are a way to group a series of statements together.
- Functions are used to perform a specific task.
- Functions can be reused.
- Functions can be passed as arguments to other functions.
- Functions can return a value.
- Functions can be assigned to variables.

Functions in JavaScript are defined with the `function` keyword.

Syntax

```
function functionName(param1, param2) {
  // code block
}
```

Understanding function parameters

- To complete the statement, A function is a block of code that performs a specific task, `when called`.
- In order to call a function, we use the function name followed by parentheses `()`.
- The parameters of a function are the values that are passed to the function when it is called.
- The parameters are separated by commas `,`.
- The parameters are optional, and function can be called without any parameters.

Example

```
function sayHello(name) {
  console.log("Hello " + name);
}
sayHello("John");
```

- The function `sayHello` is defined with the `function` keyword, followed by the name of the function.
- The function has one parameter, `name`, which is a string.
- The function logs out a hello message, followed by the name of the person.
- The function is called with the `sayHello` function name followed by parentheses `()`, followed by the name of the person we want to say hello to.
- The message is logged out to the console.

Parameters

- There can be multiple parameters in a function.

```
function sayHello(name, age) {
  console.log("Hello " + name + " you are " + age + " years old.");
}
sayHello("John", 30);
```

The return statement

- The return statement is optional in a function.
- It is used to return a value from a function.
- It is used when we need to make some calculations and return the result.

Syntax

```
function add(num1, num2) {
  return num1 + num2;
}
```

- Here the function add returns the sum of two numbers.

Working with return statements

- The return statement is optional in a function, but if it is used, it must be the last statement in the function because it will stop the function.

The alert, prompt and confirm functions

- The alert function displays a message to the user and waits for the user to press OK.
- The prompt function displays a message to the user and waits for the user to enter a value.
- The confirm function displays a message to the user and waits for the user to press OK or Cancel.

Alert

```
alert("Hello World!");
```

Prompt

```
var name = prompt("What is your name?");
alert("Hello " + name);
```

Confirm

```
var answer = confirm("Do you like JavaScript?");
if (answer) {
  alert("You said yes!");
} else {
  alert("You said no!");
}
```

Objects in JavaScript

- Objects are collections of properties, which are named values that are associated with the object.
- Objects are similar to arrays, except that objects are not ordered.
- Objects are useful for storing data.

Object Syntax

```
var person = {
  name: "John",
  age: 30,
};
```

- Here we have created an object named person, with two properties: name and age.
- The name property has a value of "John".
- The age property has a value of 30.

Accessing Object Properties

- To access the value of a property, we use dot notation.

```
var person = {
  name: "Mike",
  age: 33,
};
console.log(person.name);
console.log(person.age);
```

- The other way to access the value of a property is by using bracket notation.

```
var person = {
  name: "John",
  age: 30,
};
console.log(person["name"]);
```

- To find out the size of the object, we can use the `length` property.

```
var person = {
  name: "John",
  age: 30,
};
console.log(person.length);
```

- This will print out the number of properties in the object, here being 2.

Object Methods

- The objects can also include methods as part of their definition.
- Methods are functions that are associated with an object as properties.

```
var person = {
  name: "John",
  age: 30,
  sayHello: function () {
    alert("Hello " + this.name);
  },
};
person.sayHello();
```

- The `sayHello` method is a method of the person object and can be called on the person object using the dot notation and the `()` brackets since it is a function basically.

The Object Constructor

- Previously we created objects using the object literal syntax.

```
var person = {
  name: "John",
  age: 30,
};
```

- This allows us to create a single object only.
- In order to create multiple objects, we need to use the constructor function, so we can define object types.

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
```

- This is a standard way of creating constructor function, here the this keyword is used to refer to the object that is being created.
- this is a reserved keyword in JavaScript, so we can't use it as a variable name.
- The constructor function is called with the new keyword, so we can create multiple objects.

```
var person1 = new Person("John", 30);
var person2 = new Person("Sara", 25);
console.log(person1, person2);
```

- This creates two objects, each with their own properties as defined in the parameters.
- The new keyword is used to create a new object, and the constructor function is called with the new keyword.

Methords

- Methords are functions that are attached to objects properties.
- We use the following syntax to create a methord:

```
methordName = function () {
  // code goes here
};
```

- Access an object's methord:

```
objectName.methordName();
```

- A methord is a function belonging to an object, it can be referenced using the this keyword.
- `this` keyword is used as a reference to the object that the methord belongs to
- Defining methords is done inside the constructor function.

```
function person(name, age) {
  this.name = name;
  this.age = age;
  this.sayName = function () {
    console.log(this.name);
  };
}

var p = new person("John", 30);
p.sayName();
```

Methords can be defined outside of the constructor function

```
function person(name, age) {
  this.name = name;
  this.age = age;
  this.yearOfBirth = bornYear;
}
function bornYear() {
  return 2016 - this.age;
}
```

- As you can see, we have assigned the object's yearOfBirth property to the bornYear function.

- The this keyword is used to access the age property of the object, which is going to call the method.
- Note that it is not nessasary to write the function's parenthesis when calling the methord.
- We call the methord by the property name.

```
function person(name, age) {
  this.name = name;
  this.age = age;
  this.yearOfBirth = bornYear;
}
function bornYear() {
  return 2016 - this.age;
}

var p = new person("A", 22);

document.write(p.yearOfBirth());
```

- Note : In order to use the objects properties within a funtion use the this keyword

Arrays in JavaScript

- Arrays store multiple values in a single variable.

Syntax

```
var names = new Array("John", "Mark", "Jane");
```

- The syntax declared three person names in an array.

Accessing Elements

```
var names = new Array("John", "Mark", "Jane");
names[0]; // John
names[1]; // Mark
names[2]; // Jane
```

- The arrays are zero-based, so the first element is at index 0.
- If we want to access the second element, we would use the index 1.
- If we access some index out of the array's range, we will get undefined.

Other ways to create arrays

- We can declare an array with a specific length and then add values to it.

```
var names = new Array(3);
names[0] = "John";
names[1] = "Mark";
names[2] = "Jane";
```

- Arrays are special type of object, an array uses numbers to access its elements, an object uses names to access its elements.
- In js arrays are dynamic, so we can define an array and not pass any arguments with the Array() constructor, you can add dynamically.

Array Literal

- For improved readability, we can use array literals.

```
var names = ["John", "Mark", "Jane"];
```

- You can access and modify the elements of the array using the index number, as you did before.
- The array literal syntax is the recommended way to declare arrays.

Array Properties

- Array has a bunch of usefull built in properties and methods.

The length property

- An arrays length returns the number of its elements.

```
var array = [1, 2, 3];
console.log(array.length); // 3
```

- The length property is read only.
- All indexes in an array are zero based.
- The length property is always 1 greater than the highest index.

Quick Quiz

- Q. Array has a length property because it is an?

`a function` `an object` `a variable`

> Ans : `an object`

Combining Arrays

- JavaScripts concat() methord allows you to combine two or more arrays and create an entirely new array.

```
var c1 = [1, 2, 3];
var c2 = [4, 5, 6];
var c3 = c1.concat(c2); // [1, 2, 3, 4, 5, 6]
```

- concat does not affect the original arrays.
- concat returns a new array.

Quick Quiz

- Q. concat() takes two or more arrays and ?

> Ans : `returns a new array` `does not affect the original arrays`

Associative Arrays

- While many programming languages support arrays with named indexes, JavaScript does not.
- These are called associative arrays, in JavaScript, these are rather treated as objects.
- So, in JavaScript, when we use a named array syntax, we are actually using an object.

Example

- Representing a person.

```
var person = []; // empty array
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 50;
console.log(person); // {firstName: "John", lastName: "Doe", age: 50}
```

- Here the person array gets treated as an object.
- Hence the standard array methods are not available, and yeild different results.
- Like person.length will return 0. Which is an array property and not an object property.
- But person.hasOwnProperty("firstName") will return true. Which is a valid property for object.

Quick Quiz

- Q. In associative arrays, index numbers are replaced with ?

`constants` `strings` `variable names` `functions`

- Q. In order to use associative arrays, we need to which syntax ?

`[]` `{}` `()`

The Math Object

- The Math object has a bunch of useful methods and properties.

Property	Description
PI	The value of PI, the ratio of a circle's circumference to its diameter.
E	The base of the natural logarithm, the natural logarithm of 1.
LN2	The natural logarithm of 2.
LN10	The natural logarithm of 10.
LOG2E	The base 2 logarithm of E.
LOG10E	The base 10 logarithm of E.
SQRT1_2	The square root of 1/2.
SQRT2	The square root of 2.

Example

```
console.log(Math.PI); // 3.141592653589793
console.log(Math.E); // 2.718281828459045
console.log(Math.LN2); // 0.6931471805599453
console.log(Math.LN10); // 2.302585092994046
console.log(Math.LOG2E); // 1.4426950408889634
console.log(Math.LOG10E); // 0.4342944819032518
console.log(Math.SQRT1_2); // 0.7071067811865476
console.log(Math.SQRT2); // 1.4142135623730951
```

- Math object has no constructor, we need not to create an object for it.

Math object methords

- Math object contains a bunch of methods used for mathematical operations/calculations.

Methord	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine of a number.
asin()	Returns the arcsine of a number.
atan()	Returns the arctangent of a number.
atan2()	Returns the arctangent of the quotient of its arguments.
ceil()	Returns the smallest integer greater than or equal to a number.

Methord	Description
`cos()`	Returns the cosine of a number.
`exp()`	Returns e to the power of a number.
`floor()`	Returns the largest integer less than or equal to a number.
`log()`	Returns the natural logarithm (base E) of a number.
`max()`	Returns the largest of zero or more numbers.
`min()`	Returns the smallest of zero or more numbers.
`pow()`	Returns a number raised to a power.
`random()`	Returns a random number between 0 and 1.
`round()`	Rounds a number to the nearest integer.
`sin()`	Returns the sine of a number.
`sqrt()`	Returns the square root of a number.
`tan()`	Returns the tangent of a number.

- For example to get a random number between 0 and 1, we can use the following code:

```
var randomNumber = Math.random();
console.log(randomNumber); // 0.9888888888888889
```

- Intrestingly enough, the random number is not a whole number, so to get a random number between (0-9) or (1-10), we can use the following code:

```
var randomNumber = Math.random() * 10;
console.log(randomNumber); // 9.999999999999998
// to get a whole number (0-9), we can use the following code
var randomNumber = Math.floor(Math.random() * 10); // 0, 1, 2, 3, 4, 5, 6,
7, 8, 9
// or ceil method (1-10)
var randomNumber = Math.ceil(Math.random() * 10); // 1, 2, 3, 4, 5, 6, 7,
8, 9, 10
```

Quick quiz

- Q. What is the difference between Math.random() and Math.floor(Math.random() * 10) ?

  `Math.random()` returns a random number between 0 and 1.
  `Math.floor(Math.random() * 10)` returns a random whole number between 0 and 10.

- Q. Which methord is used to find square root of a number?

```
Math.sqrt()
```

- Q. What will be the result of `Math.sqrt(9)`?

```
3
```

setInterval()

- setInterval() is a method of the window object (browser) that executes a function or a string of JavaScript code at specified intervals (in milliseconds).
- It will continue to execute the function or the JavaScript code until clearInterval() is called on the interval ID.

Example

```
function myAlert() {
  alert("Hello World!");
}
setInterval(myAlert, 1000);
```

- This will call the function myAlert() every second.
- Here 1000 milliseconds is 1 second.
- Note : Here the myAlert is passed without the parenthesis in the setInterval() method.

Quick Quiz

- Fill in the blanks to call the calc() every 3 seconds:

```
function calc() {
  console.log(Math.random() * 10);
}

setInterval(___, ___);
```

The Date object

- The Date object is a part of the JavaScript language.
- It is used to represent a date and time.
- It is used to get the current date and time.
- It consists of methods to get the current date, time, year and so on.
- Using the new Date(), we can create a new Date object with the current date and time.

Creating a new Date object

```
var today = new Date();
```

- The other way to create a new Date object is to pass the date and time as arguments to the new Date() method.

```
var today = new Date(year, month, day, hours, minutes, seconds,
milliseconds);
```

- Js Dates are calculated from 01 January, 1970. 00:00:00 (UTC)
- One day contains 86,400,000 milliseconds.
- One hour contains 3,600,000 milliseconds.
- One minute contains 60,000 milliseconds.
- One second contains 1,000 milliseconds.

Example

```
// Fri Jan 02 1970 00:00:00
var d1 = new Date(86400000);

// Sat Jan 03 1970 00:00:00
var d2 = new Date(86400000 * 2);
```

- JavaScript counts months from 0 to 11 (January to December).
- The computer time is ticking but the date object does not.

Quick Quiz

- Q. What information results from creating a Date object with the new Date() method?

> A. The current date and time.
> B. The current date.
> C. The current time.
> D. The current year.

Date Methods

- When a Date object is created, a number of methods make it possible to get information about the date and time.

Method	Description
getDate()	Returns the day of the month (from 1 to 31)
getDay()	Returns the day of the week (from 0 to 6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0 to 23)
getMilliseconds()	Returns the milliseconds (from 0 to 999)
getMinutes()	Returns the minutes (from 0 to 59)
getMonth()	Returns the month (from 0 to 11)

Method	Description
getSeconds()	Returns the seconds (from 0 to 59)
getTime()	Returns the number of milliseconds since January 1, 1970
getTimezoneOffset()	Returns the time-zone offset from UTC
getUTCDate()	Returns the day of the month in universal time
getUTCDay()	Returns the day of the week in universal time
getUTCFullYear()	Returns the year in universal time
getUTCHours()	Returns the hour in universal time
getUTCMilliseconds()	Returns the milliseconds in universal time
getUTCMinutes()	Returns the minutes in universal time
getUTCMonth()	Returns the month in universal time
getUTCSeconds()	Returns the seconds in universal time
setDate()	Sets the day of the month (from 1 to 31)
setFullYear()	Sets the year (four digits)
setHours()	Sets the hour (from 0 to 23)
setMilliseconds()	Sets the milliseconds (from 0 to 999)
setMinutes()	Sets the minutes (from 0 to 59)
setMonth()	Sets the month (from 0 to 11)
setSeconds()	Sets the seconds (from 0 to 59)
setTime()	Sets the number of milliseconds since January 1, 1970
setUTCDate()	Sets the day of the month in universal time
setUTCFullYear()	Sets the year in universal time
setUTCHours()	Sets the hour in universal time
setUTCMilliseconds()	Sets the milliseconds in universal time
setUTCMinutes()	Sets the minutes in universal time
setUTCMonth()	Sets the month in universal time
setUTCSeconds()	Sets the seconds in universal time
toDateString()	Returns the date as a string
toISOString()	Returns the date as a string in ISO format
toJSON()	Returns the date as a string in JSON format
toLocaleDateString()	Returns the date as a string in the current locale

Method	Description
toLocaleString()	Returns the date as a string in the current locale
toLocaleTimeString()	Returns the time as a string in the current locale
toString()	Returns the date as a string
toTimeString()	Returns the time as a string
toUTCString()	Returns the date as a string in UTC

... and many more.

For Example

```
var today = new Date();
today.getDate();
today.getDay();
today.getFullYear();
today.getHours();
```

- Lets say we want to develop a function printTime() that prints the current time to browser every second.

```
function printTime() {
  var d = new Date();
  var hours = d.getHours();
  var minutes = d.getMinutes();
  var seconds = d.getSeconds();
  document.body.innerHTML = hours + ":" + minutes + ":" + seconds;
}
setInterval(printTime, 1000);
```

- We declared a function printTime() that prints the current time to browser every second by using setInterval() method.
- The innerHTML property of the document.body object is used to write the content of the page.

ECMAScript 6

ECMAScript is a scripting language specification created to standardize JavaScript.

The sixth edition of ECMAScript was released in June 2015. Which was ES6 later renamed to ES2015, adds a number of new features.

- It adds support for the new **class** syntax.
- It adds support for **let** and **const**.
- It adds support for **arrow functions**.
- It adds support for **generators**.
- It adds support for **default parameters**.
- It adds support for **destructuring**.
- It adds support for **spread and rest parameters**.
- It adds support for **template strings**.
- It adds support for **symbols**.
- It adds support for **object rest/spread**.
- It adds support for **async functions**.
- It adds support for **async/await**.
- It adds support for **modules**.
- It adds support for **import/export**.
- It adds support for **decorators**.
- It adds support for **proxies**.
- It adds support for **iterators**.
- It adds support for **generators**.
  and many more.

In other words, ES6 is a superset of ES5(JavaScript). The reason why ES6 became so popular is that it introduced new conventions and OOPS concepts such as classes, inheritance, etc.

Var and Let

- In ES6 we have 3 ways to declare variables.

```
var a = 10;
let b = 20;
const c = 30;
```

- The type of declaration depends on the scope of the variable.
- Scope is a fundamental concept in all programming languages that defines the usage and visibility of variables.

var & let

- Unlike var keyword, which is global scope,or locally to an entire function reguardless of block, let allows you to define a variable that is limited to the block it is defined in.

```
if (true) {
  let a = 10;
}
alert(a); // error
```

- In this case the variable is defined using block scope (let) and is being used in the outer block.
- Hence the variable is not accessible outside the block.
- To demonstrate the diffrence between var and let, let is used in the following example.

```
function test() {
  var a = 10;
  if (true) {
    let b = 20;
    console.log(a); // 10
    console.log(b); // 20
  }
  console.log(a); // 10
  console.log(b); // error
}
```

- One of the best uses of let is in loops.

```
for (let i = 0; i < 10; i++) {
  console.log(i); // 0,1,2,3,4,5,6,7,8,9
```

```
}
console.log(i); // error
```

- Since the variable is defined using block scope, it is not accessible outside the block.
- This behaviour is as it is supposed to be for loops as the iterator variable is not accessible outside the loop, and isnt supposed to be accessible outside the loop.

> Note : let is not subject to Variable Hoisting, which means that let declarations are not hoisted to the top of the scope or the current execution context.

Quiz Time

- What will be the output of the following code?

```
function func() {
  let a = 10;
  if (true) {
    let a = 20;
    console.log(a);
  }
  console.log(a);
}
func();
```

Answer: 20,10

const keyword

- const variables are like let variables, and have same scope as let, but they cannot be reassigned.
- const variables are immutable i.e. they are not allowed to change their value (reassign).

```
const a = 10;
a = 20; // error
```

- Simmilarly const declarations are not hoisted to the top of the scope or the current execution context.
- Also note that ES6 code will only run in supported browsers, and older browsers will not run ES6 code and will throw an error(syntax error).

Quick Quiz

- Fill in the blanks for appropriate keywords.

```
____ length = 10;
let sum = 0;
for(___ i = 0; i < length; i++) {
```

```
    sum += i;
  }
```

Template Literals in ES6

- Template literals are string literals that use backticks (`) instead of quotation marks (")
- Prior to ES6, you had to use concatenation to join strings together.
- Template literals allow you to use variables and expressions in your strings.

```
let name = "John";
let age = 25;
console.log("My name is " + name + " and I am " + age + " years old."); //
prior to ES6
console.log(`My name is ${name} and I am ${age} years old.`); // ES6
```

- ES6 introduced a new way to write strings, new way to output strings is using template literals.
- We use backticks (`) instead of quotation marks (") to create template literals.
- Template literals can use variables and expressions.
- Template literals are enclosed by the backticks (`) and the contents of the template literal can be separated by ${}
- Template literals can be multiline.
- Template literals can be used to output HTML.
- The ${expression} is called a template expression and has a placeholder for the result of the expression.
- To escape the backtick (`) character, you can use the backslash ().

Quic Quiz

- Fill in the blanks to complete the following code:

```
let number = 10;
let string = `The number is _{number}`;
console.log(_____);
```

Loops in ES6

- The for...in loop is used to iterate over the properties of an object.
- The for...of loop is used to iterate over the values of an iterable object, such as an array, object, or string.

For in loop example

```
let obj = {
  name: "John",
  age: 30,
};
for (let key in obj) {
  console.log(key); // name, age
}
```

- The for..in loop should not be used to iterate over an array as the JavaScript engine can not guarantee the order of the elements in the array.
- Also instead of iterating variable being number it is string so if we do math over it it will do string concatenation instead of math addition.

For of loop example

- During the for..of loop, the value of the variable is the value of the current element.
- The for...of loop works for both arrays and iterable objects.

```
// for string
let str = "Hello";
for (let char of str) {
  console.log(char); // H, e, l, l, o
}
```

- for...of loop also works on newly introduced collections, such as Map and Set.

Quick Quiz

- Fill in the blanks to complete the following for...of loop statement:

```
___ (let ch __ "javascript) {
  console.log(ch); // j, a, v, a, s, c, r, i, p, t
}
```

Functions in ECMA Script 6

- Prior to ES6, functions were not first class objects.
- In ES6, functions are first class objects.
- In ES6, functions can be used as values.
- In ES6, functions can be passed as arguments to other functions.
- In ES6, functions can be returned from other functions.
- Prior syntax:

```
function sayHello() {
  console.log("Hello");
}
sayHello();
```

- ES6 syntax:

```
const sayHello = () => {
  console.log("Hello");
};
sayHello();
```

- With arguments:
- Prior syntax:

```
function sayHello(name) {
  console.log("Hello " + name);
}
```

- ES6 syntax:

```
const sayHello = (name) => {
  console.log(`Hello ${name}`);
};
sayHello("John");
```

- New syntax is quite handy when you just need a simple function with one argument, we can skip writing braces and return keyword.

```
const greet = (name) => `Hello ${name}`;
```

- for no parameters, empty parentheses are used.

```
const greet2 = () => console.log("Hello");
```

- Syntax is quite usefull for inline functions.
- Lets take an example, we have an array and for each element we want to execute a function.

```
var arr = [1, 2, 3, 4, 5];
arr.forEach(function (element) {
  console.log(element * 2); // 2, 4, 6, 8, 10
});
```

- However in ES6 we can use arrow functions to do the same.

```
arr.forEach((element) => console.log(element * 2));
```

Quick Quiz

- Fill in the blanks to declare an arrow function that takes an array `arr` and returns odd numbers:

```
const printOdds = (___) => {
  ___.forEach((____) => {
    if (element % 2 !== 0) {
      console.log(element);
    }
  });
};
```

Default parameters in ES6

- In ES6, default parameters are added to functions right in the signature of the function.
- for example:

```
function greet(name = "John") {
  console.log(`Hello ${name}`);
}
greet(); // Hello John
greet("Sara"); // Hello Sara
```

- Here's an arrow function with default parameters:

```
const greet = (name = "John") => console.log(`Hello ${name}`);
greet(); // Hello John
greet("Sara"); // Hello Sara
```

- Default value expressions are evaluated at function call time from left to right.
- This also means that default expressions can use the values of parameters that have been passed to the function previously.

Quick Quiz

- What is the output of the following code?

```
function magic(a, b = 30) {
  return a + b;
}
console.log(magic(2));
```

> Answer: 32

Objects in ES6

- In JavaScript variables can be object data types that contain many values called properties.
- An Object can also have properties that are functions defination called methods for performing actions on the object.
- ES6 introduces shorthand notations and computed property names that make declaring objects easier.
- The new defination shorthand notation does not require : or a function keyword.
- Example:

```
let tree = {
  height: 10,
  color: "green",
  grow() {
    this.height += 10;
  },
};

tree.grow();
console.log(tree.height); // 20
```

- While creating duplicate properties the last one will be used.

```
var a = { x: 1, x: 2 };
console.log(a.x); // 2
```

- While creating duplicate properties in ES5 generated SyntaxError.

Quick Quiz

- Fill in the blanks for the code to output 50 :

```
let car = {
  speed: 30,
  accelerate() {
    ___.speed += 10;
  },
};
___.accelerate();
car.accelerate();
console.log(___.speed);
```

Computed Property Names

- With ES6 we can use computed property names to create dynamic property names.
- [ ] syntax we can create a property name dynamically.
- Example:

```
let prop = "name";
let id = "1234";
let mobile = "09631";
let user = {
  [prop]: "John",
  [`user_${id}`]: `${mobile},
};
```

Example

```
var i = 0;
var a = {
  ["foo" + ++i]: i,
  ["foo" + ++i]: i,
  ["foo" + ++i]: i,
};
console.log(a); // { foo1: 1, foo2: 2, foo3: 3 }
```

Example

```
var param = "size";
var config = {
  [param]: 12,
  ["mobile" + param.charAt(0).toUpperCase() + param.slice(1)]: 5,
};
console.log(config.mobileSize); // 5
```

- Its very useful when we want to create dynamic property names, custom based on some variable rules

Quick Quiz

- Fill in the blanks to create an object with its properties:

```
let prop = "foo";
let obj = {
  _prop]: "bar"_
  [prop.toUpperCase()]_ "BAR",
```

```
  [prop.length]: 3_
};
console.log(obj); // { foo: 'bar', FOO: 'BAR', 3: 3 }
```

Object.assign() in ES6

- Object.assign() is a new built-in function that copies the values of all enumerable own properties from one or more source objects to a target object.
- Its usefull for creating deep copies of objects.
- Example:

```
let obj1 = { a: 1, b: 2 };
let obj2 = { b: 4, c: 5 };
let obj3 = { c: 6 };
let newObj = Object.assign({}, obj1, obj2, obj3);
console.log(newObj); // { a: 1, b: 4, c: 6 }
```

- Here we are creating a new object and copying the properties of the other objects into it, the first object is the target object {}.
- Every parameter passed to Object.assign() is an object that will be copied to the target object.
- The order of the parameters is important, the first object will be copied to the target object.
- The target object will be modified.
- The simple = assignment creates a reference to the object, so the target object will be modified in mutation.
- To avoid this behavior we can use Object.assign() to create a new object and copy the properties of the other objects into it.

Quick Quiz

- What will be the output of the following code?

```
const ob1 = { a: 1, b: 2, c: 4 };
const ob2 = Object.assign({ c: 5, d: 6 }, ob1);
console.log(ob2.c, ob2.d);
```

> Answer: 4, 6 as the target object is a new object it will be modified.

Array Destructuring in ES6

- Destructuring arrays is a new feature in ES6.
- It makes possible to extract values from arrays and assign them to variables.
- Example:

```
let arr = [1, 2, 3];
let [a, b, c] = arr;
console.log(a, b, c); // 1 2 3
```

- We can also destructure arrays returning from a function.

```
function getArr() {
  return [1, 2, 3];
}
let [a, , c] = getArr();
```

- Notice that we left the second value empty, so it will be undefined.
- The destructuring syntax makes it easier to also swap values.

```
let a = 1;
let b = 2;
[a, b] = [b, a];
console.log(a, b); // 2 1
[a, b = 3] = [2];
console.log(a, b); // 2 3
```

Quick Quiz

- What is the output of the following code?

```
let names = ["John", "Jane", "Mary"];
let [Ann, Bob, Charlie] = names;
console.log(Ann, Bob, Charlie);
```

> John Jane Mary

Object Destructuring

- Destructuring objects is a new feature in ES6.
- It makes possible to extract values from objects and assign them to variables.

- Example:

```
let obj = {
  a: 1,
  b: 2,
  c: 3,
};
let { a, b, c } = obj;
console.log(a, b, c); // 1 2 3
```

- We can assign without declaration, but theres syntax to remember.

```
let a, b;
({ a, b } = { a: 1, b: 2 });
console.log(a, b); // 1 2
```

- The () with semi-colon ; is a way to assign without declaration.

ES6 Rest Parameters

- Prior to ES6, if we wanted to pass a variable number of arguments to a function, we had to use the `arguments` object, an array like object to access parameters passed to a function.

```
// for example lets write a function that array contains all the arguments
passed to it
function containsAll(arr) {
  for (let k = 1; k < arguments.length; k++) {
    let num = arguments[k];
    if (arr.indexOf(num) === -1) {
      return false;
    }
  }
  return true;
}
let x = [1, 2, 3, 4, 5];
console.log(containsAll(x, 1, 2, 3)); // true
console.log(containsAll(x, 1, 2, 3, 4)); // false
```

- In ES6, we can use the `...` syntax to pass a variable number of arguments to a function.

```
function containsAll(arr, ...nums) {
  for (let num of nums) {
    if (arr.indexOf(num) === -1) {
      return false;
    }
  }
  return true;
}
```

- The ...nums parameter here is called the rest parameter. It is an array-like object that contains all the arguments passed to the function after the first parameter.
- The ... are called the Spread Operator.

Note

- Only the last parameter can be a rest parameter. If there are no extra arguments, the rest parameter will simply be an empty array, the rest parameter will never be undefined.
- The rest parameter can only be used in function calls.

Quick Quiz

- What will be the output of the following code?

```
function magic(...nums) {
  let sum = 0;
  nums.filter((n) => n % 2 == 0).map((el) => (sum += el));
  return sum;
}
console.log(magic(1, 2, 3, 4, 5, 6));
```

- 12

Spread Operator

- The operator `...` is used to spread an array into a list of arguments.
- It is simmilar to the rest parameter, but it is used to spread an array into a list of arguments.

Spread in function calls

- It is common to pass elements of an array as arguments to a function.
- Before ES6 we used to,

```
function myFunction(a, b, c) {
  console.log(a + b + c);
}
var args = [1, 2, 3];
console.log(myFunction.apply(null, args));
```

- ES6 gives an easier way to do this using the spread operator.

```
function myFunction(a, b, c) {
  console.log(a + b + c);
}
var args = [1, 2, 3];
console.log(myFunction(...args));
```

- One more example of spread operator in function calls.

```
function myFunction(a, b, c) {
  console.log(a, b, c);
}
var args = [1, 2];
myFunction(...args, 3);
```

Spread in Array Literals

- Before ES6, we used the following syntax to add an item at middle of an array:

```
var arr = ["One", "Two", "Seven"];
arr.splice(2, 0, "Three");
arr.splice(3, 0, "Four");
arr.splice(4, 0, "Five");
arr.splice(5, 0, "Six");
console.log(arr); // ["One", "Two", "Three", "Four", "Five", "Six",
"Seven"]
```

- You can use methords such as `push`, `splice`, `concat` to add items to an array.
- But it is easier to use the spread operator to add an item at middle of an array.

```
var arr = ["One", "Two", "Seven"];
arr.splice(2, 0, ...["Three", "Four", "Five", "Six"]);
console.log(arr); // ["One", "Two", "Three", "Four", "Five", "Six",
"Seven"]
```

Spread in Object Literals

- In objects it copies the own enumerable properties from the provided object onto a new object.

```
const ob1 = {
  a: 1,
  b: 2,
  c: 3,
};
const ob2 = {
  d: 4,
  e: 5,
  f: 6,
};
const clone = { ...ob1, ...ob2 }; // { a: 1, b: 2, c: 3, d: 4, e: 5, f: 6
}
```

- Shallow cloning or merging objects is possible with another operator called `Object.assign()`.

Quiz Time

- What is the output of the following code?

```
let nums = [1, 4, 1, 5];
let all = [3, ...nums, 9, 2];
console.log(all[2]);
```

- all : [3, 1, 4, 1, 5, 9, 2]

- all[2] : 4

Classes in ES6

- In this tutorial we'll learn how to create a class that can be used to create multiple objects of the same structure.
- A class uses the keyword class and contains a constructor method for initializing.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
```

- A declared class can then be used to create multiple objects using the keyword new.

```
const person1 = new Person("John", 30);
const person2 = new Person("Sara", 25);
```

- Note : Class Declarations are not hoisted while Function Declarations are. If you try to access your class before declaring it, ReferenceError will be returned.
- You can also define a class with a class expression, where the class can be named or unnamed.
- A named class looks like this:

```
var Milind = class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
};
```

- In the unnamed class expression, a variable is simply assigned the class definition:

```
var Person = class {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
};
```

- The constructor is a special method which is used for creating and initializing an object created with a class.
- There can be only one constructor in each class.

Quick Quiz

- Fill in the blanks to complete this Point class

```
____ Point {
  ________(x, y) {
    this.x = x;
    this.y = y;
  }
}
getX() {
  return ____.x;
}
getY() {
  return ____.y;
}
```

Class Methords in ES6

- ES6 introduced a shorthand that does not require the keyword function to create a method.
- The method name is the same as the property name.
- One type of methord is prototype methord, which is available to all instances of the class.

For example, the following class has a methord called `area` which returns the value of the `height * width` properties.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  get area() {
    return this.calcArea();
  }
  calcArea() {
    return this.height * this.width;
  }
}
const square = new Rectangle(5, 5);
console.log(square.area); // 25
```

- In the code above, area is a getter, calcArea is a method.
- Getter is a methord that returns the value of a property.

- Another type of method is the static method, which cannot be called through a class instance. Static methods are often used to create utility functions for an application

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
  static distance(a, b) {
    const dx = a.x - b.x;
    const dy = a.y - b.y;
    return Math.hypot(dx, dy);
  }
}
const p1 = new Point(7, 2);
const p2 = new Point(3, 8);
console.log(Point.distance(p1, p2));
```

- As you can see, the static distance method is called directly using the class name, without an object.

Quiz Time

- Fill in the blanks to complete this class so that it outputs : `Jimmy says woof!`

```
class Dog {
  constructor(name) {
    ____.name = name;
  }
  bark() {
    console.log(this.____ + " says woof!");
  }
}
let d = new Dog("Jimmy");
d.____();
```

Inheritance in ES6

- The extends keyword is used in class declarations or class expressions to create a child of a class.
- The child inherits the properties and methods of the parent.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(this.name + " makes a noise.");
  }
```

```
}
class Dog extends Animal {
  speak() {
    console.log(this.name + " says woof!");
  }
}
let dog = new Dog("Jimmy");
dog.speak(); // Jimmy says woof!
```

- In the code above, the Dog class is a child of the Animal class, inheriting its properties and methods.
- If there is a constructor present in the subclass, it needs to first call super() before using this.
- Also, the super keyword is used to call parent's methods.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(this.name + " makes a noise.");
  }
}

class Dog extends Animal {
  speak() {
    super.speak(); // Super
    console.log(this.name + " says woof!");
  }
}

let dog = new Dog("Jimmy");
dog.speak();
// Jimmy makes a noise.
// Jimmy says woof!
```

- In the code above, the parent's speak() method is called using the super keyword.

Quiz Time

- Fill in the blanks to declare a class `Student` which inherits from the `Human` class.

```
class Human {
  constructor(name) {
    this.name = name;
  }
}
class ____ extends ____ {
  constructor(name, age) {
    ____(name, age);
    this.age = age;
```

```
    }
  }
```

ES6 Map

- A Map object can be used to hold key/value pairs. A key or value in a map can be anything (objects and primitive values).
- The syntax new Map([iterable]) creates a Map object where iterable is an array or any other iterable object whose elements are arrays (with a key/value pair each).
- An Object is similar to Map but there are important differences that make using a Map preferable in certain cases:
- 1. The keys can be any type including functions, objects, and any primitive.
- 2. You can get the size of a Map.
- 3. You can directly iterate over Map.
- 4. Performance of the Map is better in scenarios involving frequent addition and removal of key/value pairs.
- The size property returns the number of key/value pairs in a map.
  For example:

```
let map = new Map([
  ["k1", "v1"],
  ["k2", "v2"],
]);

console.log(map.size); // 2
```

Methods

- set(key, value) Adds a specified key/value pair to the map. If the specified key already exists, value corresponding to it is replaced with the specified value.
- get(key) Gets the value corresponding to a specified key in the map. If the specified key doesn't exist, undefined is returned.
- has(key) Returns true if a specified key exists in the map and false otherwise.
- delete(key) Deletes the key/value pair with a specified key from the map and returns true. Returns false if the element does not exist.
- clear() Removes all key/value pairs from map.
- keys() Returns an Iterator of keys in the map for each element.
- values() Returns an Iterator of values in the map for each element.
- entries() Returns an Iterator of array[key, value] in the map for each element.

- For example:

```
let map = new Map();
map.set("k1", "v1").set("k2", "v2");

console.log(map.get("k1")); // v1
console.log(map.has("k2")); // true

for (let kv of map.entries()) console.log(kv[0] + " : " + kv[1]);
```

- The above example demonstrates some of the ES6 Map methods.
- Map supports different data types i.e. 1 and "1" are two different keys/values.

ES6 Set

- A Set object can be used to hold unique values (no repetitions are allowed).
- A value in a set can be anything (objects and primitive values).
- The syntax new Set([iterable]) creates a Set object where iterable is an array or any other iterable object of values.
- The size property returns the number of distinct values in a set.
- For example:

```
let set = new Set([1, 2, 4, 2, 59, 9, 4, 9, 1]);

console.log(set.size); // 5
```

Methods

-add(value) Adds a new element with the given value to the Set.
-delete(value) Deletes a specified value from the set.
-has(value) Returns true if a specified value exists in the set and false otherwise.
-clear() Clears the set.
-values() Returns an Iterator of values in the set.

-For example:

```
let set = new Set();
set.add(5).add(9).add(59).add(9);

console.log(set.has(9));

for (let v of set.values()) console.log(v);
```

- The above example demonstrates some of the ES6 Set methods.
- Set supports different data types i.e. 1 and "1" are two different values.
- NaN and undefined can also be stored in Set.