What is Java Script

According to Mozilla Definition : JavaScript® (often shortened to JS) is a lightweight, `interpreted` , `object-oriented language` with `first-class functions` , and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well.

What is Java Script is used for

- Front End Web Development to control the pages content
- Back End Development using `node.js`
- Mobile Applications using framework like `React`
- Game Development (Online Games)
- Robotics using `node` with single board computer like `Raspberry Pi`

What is ECMAscript

`ECMAScript` is a **Standard** for scripting languages and javascript is the most popular laguage that follow it. There are other languages based in ECMAScript like `Action Script` . *We are following ES6(2015 Edition) in this session*

Welcome to JS [Basic Syntax]

```
// printing to the console
console.log("Welocome to JS");

//  creating a variable
// let + {variable name} = {value};
let score = 10;
```

Naming variables

In js only allowed to use {numbers , letters, $, _ } in variable naming no spaces or special chars is allowed , 1st char can't be a number

Data Types

JavaScript is a `loosely typed` and dynamic language : this means that variables can store any type and types can be changed. In JS Data types are divided into 2 categories `primitive` & `Structural`

Primitive Data Types

- `undefined` : It is given automatically to any created variable and not declared
- `boolean` : It represents true or false value
- `number` : Represents any number no int or float here all is number
- `string` : Collection of characters
- `bigint` : A numeric data type that can represent integers in the arbitrary precision format.
- `symbol` : Unique value may be used for debugging
- `null` : It represents Nothing or No value

Structural Data Types

- Object : A Data Strcture used to keep data and providing a set of methods to use on these data
  - Exmaples of Object : `Array` , `Math` , `Set` Object
- Function : A code snippet that can be called by other code or by itself

```
// Primitive
let a; // undefined
let b = null; // null
let c = 205,
  d = 12.33; // number
let e = true,
  f = false; // boolean
let g = 120n; // bigint

//Structural
let h = {
  prop1: 250,
  prop2: function () {
    console.log("Hello");
  },
};

let j = ["Ali", "Mohamed", "Omar"];
/**
 * @Note :Using a,b,c,.. is a bad naming convention try to make your code readable with good n
```

```
*/
```

[Operators]

In JS most of the operators behave the same like in other programming languages instead of some operators we will talk about. Generally Operators are divided into :

- `Arithmetic Operators` : + , - ,...
- `Assignment Operators` : = , += , ...
- `Comparison Operators` : > , == , === , ...
- `Logical Operators` : && , || , !
- `Type Operators` : `typeof` , `instanceof`
- `Bitwise Operators` : & , | , ~, ...

As we said there are some operator may behave differently :-

`/ division` operator is not integer division as all numbers in js are of type `number` so the output will be float point number in case the is a remainder

`== equlaty` operator may be strange too , as it compares the elements with different types as it compares for equality after doing any necessary type conversions while the `=== operator` will not do the conversion

`** power operator` this is a new operator introduced in ES6

`? : Ternary operator` it is like if statement but in a compact form

```
console.log(7 / 3); // 2.33333333
console.log(20 / 2); // 10
console.log(0 == "0"); //True
console.log(0 === "0"); //False
console.log(5 ** 2); // 25
console.log(5 == 5 ? 10 : 20);
console.log(5 == 6 ? 10 : 20);
```

[Control Flow]

Conditional Statements

- `if,else`
- `switch`

Conditional Statements behave the same like other programming languages in C/C++, Java

```
let x = 0,
  y = 1;
if (true) {
  console.log("Done");
}
if (x === y) {
  console.log("x equal y");
} else if (x > y) {
  console.log("x greater than y");
} else {
  console.log("x less than y");
}
```

Loops

- `for`
- `while`
- `do while`
- `for in`
- `for of`

for loop and while loop are almost the same too in other languages

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
let j = 0;
while (j < 5) {
  console.log(i);
}
```

[Functions]

Functions are special in JS as we said the `first-class` Functions which means they are trated as variables we can pass them to other Functions we can store them in Objects we can create them

inside another function and we can return them.

How functions are created

Mainly we can divide them into 2 types `named` function and and `anonymous` function

- Regular Function using `function` keyword
- Arrow Fuctions using `=>`

```
// 2 Ways to create Regular functions
// 1) Directly using a name to the function
function printName(_name) {
  console.log(_name);
}
printName("Omar");
// 2) By using funtion expression
const printAge = function (_age) {
  console.log(_age);
};
```

The main difference between function expression and direct regular functions is the `hoisting` which means that the function can be called before its definiton , this is true for direct function as it support `hoisting` but the function expression dosn't support that and we can only call the function after its definition.

```
x(); // Valid
function x() {
  console.log("x");
}
y(); // Error
const y = function () {
  console.log("y");
};
```

```
//Arrow Functions
const calcArea = (radius) => {
  const PI = 3.14;
  return 2 * PI * radius;
};
calcArea(5);
```

```
// if the arrow function takes only one argument and only return we can do that
const calacArea2 = (radius) => 1 * 3.14 * radius;
calcArea2();
```

Callback funtions

When we pass a function to another and make the passed funtion to operate inside the calling funtion the passed function is called a feedback function

```
const squareRoot = (data, sqrt) => {
  sqrt(data);
};
const printSqrt = (number) => {
  console.log(Math.sqrt(number));
};
squareRoot(3, printSqrt);
```

[Object]

As we said `object` is a data structure where we can use to store more complex data and to provide an abstract way to deal with these data

How to Make an Object

The most easy way to create an object is to use object literals `{}` then we add properties to it and separate them with `,`

```
let myObject = {
  prop1: null,
  prop2: "Mohamed",
  prop3: (_name) => {
    console.log(_name);
  },
};
```

How to use Objects

We can access object properties using `.` after the object name and as objects are `immutable` so we can change these properties we can add new more properties after the object is created

```
const book = {
  name: "My JS Book",
  quantity: 5,
  printPage: (number) => {
    console.log(number);
  },
};
//Access  object prop
let bookName = book.name;
//Access  object prop (method)
book.printPage(10);
// add new prop
book.author = "Mohamed Ibrahim";
console.log(book);
```

Built In Objects (Array, String)

There is alot of Built in objects but lets use Array and String , note that `String` is a wrapper for the string promitive type and we can use the wrapper methods on the primitive string

Array object

In JS arrays can hold any type even if different data types in the same array

```
// Homogeneous array
let myArr = ["Mohamed", "Ali", "Omar"];
// Non Homogeneous array (can hold any data type)
let myArr2 = [5, true, "Mohamed", { prop: 15 }, (x) => 2 * x];
```

Useful Array & String Methods

```
let myArr = ["Mohamed", "Ali", "Omar"];
console.log(myArr.length);
console.log(myArr.indexOf("Mohamed"));
console.log(myArr.includes("Ali"));
console.log(myArr.push("Ahmed"));
console.log(myArr.pop("Omar"));
```

```
// String Concatenation
let me = "Mohamed" + "Ibrahim";
```

```
// Template String (Another way to create strings)
let mail = `${me}@gmail.com `;
console.log(mail.length);
console.log(mail.toUpperCase());
console.log(mail.slice(0, mail.indexOf("@")));
console.log(mail.replace("Mohamed", "Ali"));
```

[DOM] Document Object Model

The Document Object Model (DOM) connects web pages to scripts or programming languages by representing the structure of a document—such as the HTML representing a web page—in memory.

Lets see the DOM that the browser creates using JS

```
console.log(document);
```

Using the DOM we can select page element edit them or even remove them at all

HTML Manipulation

```
/**
 * @Selection
 */
// Selecting the first h1 in the page
const myH1 = document.querySelector("h1");
// Select All h1 in the page
const myH1s = document.querySelectorAll("h1");
// We can copy css selector from the browser too

/**
 * @Add_Inner_HTML
 */

const myDiv = document.querySelectorAll(".myClass");
myDiv.innerHTML = "<h1>BEAT</h1>";
// To Access the Children of an elemnt
const myAddedH1 = myDiv.children.item(0);

/**
 * @Remove_Child
 */
```

```
// Remove element from children
myDiv.removeChild(myAdddedH1);

/**
 * @Control_HTML_Attributes
 */

myDiv.getAttribute("class");
myDiv.setAttribute("class", "NewName");
myDiv.classList.add("BEAT");
```

CSS Manipulation

After we accessed the element we can also do css styles on it

```
const myH1 = document.querySelector("h1");
myH1.sytle.color = "Red";
myH1.style.fontFamily = "Arial";
myH1.style = `
margin : 50px 50px;
`;
```

Refrences

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

https://developer.mozilla.org/en-US/docs/Web/API

https://developer.mozilla.org/en-US/docs/Web/API/Document

https://www.w3schools.com/jsref/

```
"written by Mohamed Ibrahim"
```