Variables and Data Types:

Declaring Variables using var, let, and const:

var: The traditional way of declaring variables. It has function scope.

```
var x = 5;
```

let: Introduced in ES6. It has block scope.

```
let y = 10;
```

const: Similar to let, but the value cannot be reassigned after declaration.

```
const z = 15;
```

Data Types:

String: Represents a sequence of characters.

```
let greeting = "Hello, World!";
```

Number: Represents numeric values.

```
let age = 25;
```

Boolean: Represents either true or false.

```
let isStudent = true;
```

Null: Represents the intentional absence of any object value.

```
let nullValue = null;
```

Undefined: Represents a variable that has been declared but not assigned a value.

```
let undefinedValue;
```

Operators:

Arithmetic Operators:

Addition (+)
Subtraction (-)
Multiplication (*)
Division (/)

```
let result = 10 + 5; // 15
```

Comparison Operators:

Equal (==)
Strict Equal (===)
Not Equal (!=)
Strict Not Equal (!==)

```
let isEqual = (5 === "5"); // false
```

Logical Operators:

AND (&&)
OR (||)
NOT (!)

```
let isTrue = (true && false); // false
```

Control Flow:

Conditional Statements:

if, else if, else

```
let num = 10;


if (num > 0) {
console.log("Positive");
} else if (num < 0) {
console.log("Negative");
} else {
console.log("Zero");
}
```

Switch Statements:

Used for multiple conditions.

```
let day = "Monday";


switch (day) {
case "Monday":
console.log("It's Monday!");
break;
case "Tuesday":
console.log("It's Tuesday!");
break;
// ... other cases ...
default:
console.log("It's another day.");
}
```

Looping:

for: Execute a block of code a specified number of times.

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

while: Execute a block of code while a specified condition is true.

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

do-while: Similar to while, but the condition is checked after the block is executed.

```
let j = 0;

do {

console.log(j);

j++;

} while (j < 5);
```

Functions:

Declaring and Invoking Functions:

Function Declaration:

```
function add(a, b) {

return a + b;

}
```

Function Expression:

```
let multiply = function(x, y) {

return x * y;

};
```

Arrow Function (ES6):

```
let subtract = (p, q) => p - q;
```

Parameters and Return Values:

Parameters: Values passed into a function.

```
function greet(name) {

console.log("Hello, " + name + "!");

}
```

Return Values: Values returned by a function.

```
function square(num) {

return num * num;

}
```

Arrays:

Creating Arrays:

An ordered list of values.

```
let numbers = [1, 2, 3, 4, 5];
```

Accessing and Modifying Array Elements:

Accessing Elements:

```
let firstElement = numbers[0]; // 1
```

Modifying Elements:

```
numbers[2] = 10; // [1, 2, 10, 4, 5]
```

Array Methods:

push(): Adds elements to the end.
pop(): Removes the last element.
shift(): Removes the first element.
unshift(): Adds elements to the beginning.

```
numbers.push(6); // [1, 2, 10, 4, 5, 6]
```

Objects:

Creating Objects:

A collection of key-value pairs.

```
let person = {
```

```
name: "John",

age: 30,

isStudent: false

};
```

Accessing and Modifying Object Properties:

Accessing Properties:

```
let personName = person.name; // "John"
```

Modifying Properties:

```
person.age = 31;
```

Object Methods:

Functions inside an object.

```
let car = {

brand: "Toyota",

start: function() {

console.log("Car started!");

}

};


car.start();
```

Events:

Handling Events in the Browser:

Events are user interactions with the web page.

```
<button onclick="myFunction()">Click me</button>
```

Event Listeners:

A more flexible way to handle events.

```
let button = document.getElementById("myButton");
button.addEventListener("click", function() {
console.log("Button clicked!");
});
```

DOM Manipulation:

Selecting and Modifying HTML Elements using JavaScript:

Accessing and modifying elements on the webpage.

```
let paragraph = document.getElementById("myParagraph");
paragraph.innerText = "New text!";
```

Changing Styles and Content:

Modifying styles and content of HTML elements.

```
paragraph.style.color = "blue";
```

Asynchronous JavaScript:

setTimeout and setInterval:

setTimeout: Executes a function after a specified delay.
setInterval: Repeatedly executes a function at a specified interval.

```
setTimeout(function() {
console.log("Delayed function");
}, 2000);
```

Callback Functions:

Functions passed as arguments to other functions.

```
function fetchData(callback) {
```

```
// ... asynchronous operation ...
callback(data);
}
```

Error Handling:

try, catch, finally Statements:

Used to handle exceptions in JavaScript.

```
try {
// code that might throw an exception
} catch (error) {
// handle the exception
} finally {
// code to be executed regardless of the try/catch result
}
```