Mobile Computing : Lecture-2

Created by	Ahmed Bashir
Email	ahmed.bashir@gift.edu.pk

Introduction to JavaScript

JavaScript is a programming language aka the **language of web** used for front-end and back-end development, as well as for creating mobile apps. JavaScript is essential for many of the interactive features we see on the internet today.

Some of the key features of JavaScript are:

1. Interactivity: JavaScript allows for the creation of interactive web pages and applications, enabling features like drop-down menus, autocomplete, and other interactive elements.
2. Front-end and back-end development: JavaScript can be used for both front-end and back-end development, making it a versatile language for building web applications.
3. Mobile app development: JavaScript can be used in conjunction with frameworks like React to develop mobile applications for iOS and Android.
4. Large ecosystem: JavaScript has a large and active developer community, with many libraries and frameworks available to extend its capabilities.
5. Asynchronous programming: JavaScript supports asynchronous programming, allowing for non-blocking code execution and improving the performance of web applications.
6. Functional programming: JavaScript supports functional programming paradigms, making it a powerful tool for building complex applications.
7. Dynamic typing: JavaScript is dynamically typed, meaning that variables can change their data type on the fly during runtime, making it a flexible language.

Why JavaScript?

JavaScript is a language that builds interactivity into web pages and is literally the language of the web. It is used in almost every website in some form or another. The reason for this is the fact that since its inception in 1995, JavaScript has been the main way to interact with web pages on the client-side.

One of the main reasons developers use JavaScript is because of how easy it is to use. It is considered one of the more accessible programming languages, and there are many resources available for help and guidance. Moreover, JavaScript is used on both the client-side and server-side of web development, making it a versatile language. It is not only restricted to the web but you can also build mobile applications using JavaScript.

Another reason to learn and use JavaScript is its popularity and demand in the job market. Countless job postings appear for JavaScript developers every day, and its skills are in high demand.

Writing your first JS code

To get started with JavaScript, all you need is a browser. We will be using Google Chrome.

- Open the browser
- Right click on the current active webpage and click the `Inspect` option or use the shortcut key `ctrl + shift + I` or `ctrl + shift + J`

This will open the Developer Tools and then you can click on the Console tab to open the console, or alternatively, pressing the **ESC** key will toggle on and off the console regardless of the currently active Developer Tools panel.

You can type any JavaScript command you like into the DevTools console.

If you need to type multiple lines of code before you run them, make sure to press the SHIFT + ENTER shortcut key to get onto the next line.

Notice the distinction between pressing the **ENTER** key to run the JavaScript code you've typed, versus pressing the **SHIFT + ENTER** shortcut to move onto the next line of code (rather than running the code you've already typed up).

Variables

Variables are an essential component of programming languages, including JavaScript. They are used to store and manipulate data in a program. In JavaScript, variables can be declared using the `var`, `let`, or `const` keywords, each of which has its own purpose and scope.

var

The `var` keyword was the initial and traditional way to declare variables in JavaScript. It is still used in some legacy code, but has been replaced in most cases by the `let` and `const` keywords. A variable declared with `var` is function-scoped, which means it is accessible throughout the entire function in which it is declared, including any nested functions.

```
function exampleFunction() {
  var x = 5;
  if (true) {
    var y = 10;
    console.log(x, y); // 5 10
  }
  console.log(x, y); // 5 10
}
```

let

The `let` keyword was introduced in ES6 (ECMAScript 2015) and is now the preferred way to declare variables in JavaScript. It is block-scoped, which means that a variable declared with `let` inside a block (such as an `if` statement or a `for` loop) is not accessible outside of that block. This allows for more precise control over the scope of variables.

```
function exampleFunction() {
  let x = 5;
  if (true) {
    let y = 10;
    console.log(x, y); // 5 10
  }
  console.log(x, y); // ReferenceError: y is not defined
}
```

const

The `const` keyword is used to declare constants in JavaScript. Once a value is assigned to a constant, it cannot be changed. This makes `const` useful for values that should not be modified, such as mathematical constants or URLs.

```
const PI = 3.14;
const URL = "<https://www.example.com>";
```

Like `let`, `const` is also block-scoped.

```
function exampleFunction() {
  const x = 5;
  if (true) {
    const y = 10;
    console.log(x, y); // 5 10
  }
  console.log(x, y); // ReferenceError: y is not defined
}
```

It is important to note that while the value of a constant cannot be changed, the object it points to can still be modified.

```
const person = { name: "John" };
person.name = "Jane"; // this is allowed
person = { name: "Jane" }; // this is not allowed
```

Examples

Here are some examples of declaring variables using `var`, `let`, and `const`.

```
// declaring variables with var
var firstName = "John";
var lastName = "Doe";
var age = 30;
var isEmployed = true;
```

```
// declaring variables with let
let firstName = "John";
let lastName = "Doe";
let age = 30;
let isEmployed = true;

// declaring constants
const PI = 3.14;
const URL = "<https://www.example.com>";
const person = { name: "John" };
```

In summary, variables are a crucial part of programming with JavaScript. Understanding the differences between `var`, `let`, and `const` can help you write more efficient and effective code.

Mathematical Operators in JavaScript

JavaScript supports a range of mathematical operators that can be used to perform calculations on numeric values. Some of the most commonly used operators are:

- `+` Addition - adds two values together
- Subtraction - subtracts one value from another
- Multiplication - multiplies two values together
- `/` Division - divides one value by another
- `%` Modulus - returns the remainder of a division operation
- `++` Increment - adds one to a value
- `-` Decrement - subtracts one from a value

Here are some examples of using mathematical operators in JavaScript:

```
// Addition
let x = 5;
let y = 10;
let z = x + y; // z is 15

// Subtraction
let a = 20;
let b = 5;
let c = a - b; // c is 15

// Multiplication
```

```
let p = 7;
let q = 3;
let r = p * q; // r is 21

// Division
let m = 50;
let n = 5;
let o = m / n; // o is 10

// Modulus
let s = 15;
let t = 4;
let u = s % t; // u is 3

// Increment
let v = 5;
v++; // v is now 6

// Decrement
let w = 10;
w--; // w is now 9
```

Relational Operators in JavaScript

Relational operators are used to compare values in JavaScript. They return a Boolean value, either `true` or `false`, depending on the comparison result. The most commonly used relational operators are:

- `==` Equal to
- `!=` Not equal to
- `<` Less than
- `>` Greater than
- `<=` Less than or equal to
- `>=` Greater than or equal to

Here are some examples of using relational operators in JavaScript:

```
// Equal to
let x = 5;
let y = "5";
console.log(x == y); // true

// Not equal to
let a = 10;
let b = "5";
console.log(a != b); // true

// Less than
let c = 5;
let d = 10;
console.log(c < d); // true

// Greater than
let e = 15;
let f = 10;
console.log(e > f); // true

// Less than or equal to
let g = 5;
let h = 5;
console.log(g <= h); // true

// Greater than or equal to
let i = 10;
let j = 5;
console.log(i >= j); // true
```

It is important to note that the `==` operator performs type coercion, which means that it will try to convert the operands to the same type before comparing them. This can sometimes lead to unexpected results, so it is generally recommended to use the `===` operator instead, which performs a strict comparison without type coercion.

```
// Type coercion with ==
let k = 5;
let l = "5";
console.log(k == l); // true

// Strict comparison with ===
let m = 5;
let n = "5";
console.log(m === n); // false
```

Logical Operators in JavaScript

Logical operators are used to combine or modify Boolean values in JavaScript. There are three logical operators: `&&` (and), `||` (or), and `!` (not).

&& (and)

The `&&` operator returns `true` if both operands are true, and `false` otherwise. It is often used in conditional statements to test multiple conditions at once.

```
let x = 5;
let y = 10;
if (x > 0 && y > 0) {
  console.log("Both x and y are positive");
}
```

|| (or)

The `||` operator returns `true` if either operand is true, and `false` otherwise. It is often used in conditional statements to provide a fallback option if the first condition fails.

```
let x = 5;
let y = 10;
if (x > 0 || y > 0) {
  console.log("Either x or y is positive");
}
```

! (not)

The `!` operator returns the opposite of the Boolean value of its operand. It is often used to invert a condition.

```
let x = 5;
if (!(x > 10)) {
  console.log("x is not greater than 10");
}
```

It is important to note that logical operators in JavaScript use short-circuit evaluation, which means that the second operand of a logical expression may not be evaluated if the first operand is sufficient to determine the result. For example:

```
let x = 5;
let y = null;
if (x > 0 && y.someProperty) {
  // This code will throw an error, because y is null and does not have a property named
 "someProperty"
}
```

In this example, the `y.someProperty` expression is not evaluated, because the `x > 0` expression is already `false`.

Conditional Statements in JavaScript

Conditional statements are used in JavaScript to execute certain code blocks only if certain conditions are met. The most commonly used conditional statements in JavaScript are the `if` statement and the `switch` statement.

if Statement

The `if` statement is used to execute a block of code if a specified condition is true. Here is an example of an `if` statement in JavaScript:

```
let x = 5;
if (x > 0) {
  console.log("x is positive");
}
```

In this example, the code inside the curly braces will only be executed if the condition inside the parentheses is true. If `x` is greater than 0, the console will log "x is positive".

The `if` statement can also be combined with an `else` statement to execute a block of code if the condition is false. Here is an example:

```
let x = -5;
if (x > 0) {
  console.log("x is positive");
} else {
  console.log("x is negative or zero");
}
```

In this example, if `x` is greater than 0, the console will log "x is positive". If `x` is less than or equal to 0, the console will log "x is negative or zero".

The `if` statement can also be combined with an `else if` statement to test multiple conditions. Here is an example:

```
let x = 0;
if (x > 0) {
  console.log("x is positive");
} else if (x < 0) {
  console.log("x is negative");
} else {
  console.log("x is zero");
}
```

In this example, if `x` is greater than 0, the console will log "x is positive". If `x` is less than 0, the console will log "x is negative". If `x` is equal to 0, the console will log "x is zero".

switch Statement

The `switch` statement is used to execute different code blocks based on different conditions. Here is an example of a `switch` statement in JavaScript:

```
let day = "Monday";
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;
  default:
    console.log("Today is not Monday, Tuesday, or Wednesday");
    break;
}
```

In this example, the `switch` statement checks the value of the `day` variable and executes the corresponding code block. If `day` is "Monday", the console will log "Today is Monday". If `day` is "Tuesday", the console will log "Today is Tuesday". If `day` is "Wednesday", the console will log "Today is Wednesday". If `day` is any other value, the console will log "Today is not Monday, Tuesday, or Wednesday".

It is important to note that each code block in a `switch` statement must end with a `break` statement. This is because if a `break` statement is not included, the code will continue to execute the code blocks below the matching code block, even if the conditions for those code blocks are not met.

Loops in JavaScript

Loops are used in JavaScript to execute a block of code repeatedly for a specified number of times or until a specified condition is met. The most commonly used loops in JavaScript are the `for` loop, and the `while` loop.

for Loop

The `for` loop is used to execute a block of code a specified number of times. Here is an example of a `for` loop in JavaScript:

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

In this example, the `for` loop will execute 10 times, with the variable `i` taking on values from 0 to 9. Each time the loop executes, the value of `i` is printed to the console.

The `for` loop consists of three parts: an initialization expression (`let i = 0`), a condition (`i < 10`), and an update expression (`i++`). The initialization expression is executed before the loop starts, the condition is checked before each iteration, and the update expression is executed after each iteration.

while Loop

The `while` loop is used to execute a block of code while a specified condition is true. Here is an example of a `while` loop in JavaScript:

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

In this example, the `while` loop will execute as long as the value of `i` is less than 10. Each time the loop executes, the value of `i` is printed to the console and then incremented.

The `while` loop consists of a single condition that is checked before each iteration. If the condition is true, the loop executes. If the condition is false, the loop stops.

JavaScript Arrays

An array is a data structure used to store a collection of values. In JavaScript, arrays can be created using the `[]` notation or the `Array()` constructor. Here are some examples:

```
// creating an array with the [] notation
let fruits = ["apple", "banana", "orange"];

// creating an array with the Array() constructor
let numbers = new Array(1, 2, 3, 4, 5);
```

Accessing Array Elements

Array elements can be accessed using their index. In JavaScript, array indices start at 0. Here is an example:

```
let fruits = ["apple", "banana", "orange"];
console.log(fruits[0]); // "apple"
console.log(fruits[1]); // "banana"
console.log(fruits[2]); // "orange"
```

Modifying Array Elements

Array elements can be modified by assigning a new value to their index. Here is an example:

```
let fruits = ["apple", "banana", "orange"];
fruits[1] = "kiwi";
console.log(fruits); // ["apple", "kiwi", "orange"]
```

Array Methods

JavaScript provides many built-in methods for working with arrays. Here are some of the most commonly used methods:

- `push()` adds one or more elements to the end of an array
- `pop()` removes the last element from an array and returns it
- `shift()` removes the first element from an array and returns it
- `unshift()` adds one or more elements to the beginning of an array
- `splice()` adds or removes elements from an array at a specified index
- `concat()` joins two or more arrays and returns a new array
- `slice()` returns a new array containing a portion of an existing array
- `sort()` sorts the elements of an array
- `reverse()` reverses the order of the elements in an array

Here are some examples of using array methods:

```
let fruits = ["apple", "banana", "orange"];

// push() and pop()
fruits.push("kiwi");
console.log(fruits); // ["apple", "banana", "orange", "kiwi"]
let lastFruit = fruits.pop();
console.log(lastFruit); // "kiwi"
console.log(fruits); // ["apple", "banana", "orange"]

// shift() and unshift()
let firstFruit = fruits.shift();
console.log(firstFruit); // "apple"
console.log(fruits); // ["banana", "orange"]
fruits.unshift("kiwi");
console.log(fruits); // ["kiwi", "banana", "orange"]

// splice()
fruits.splice(1, 1, "peach", "pear");
console.log(fruits); // ["kiwi", "peach", "pear", "orange"]

// concat()
let moreFruits = ["grape", "mango"];
let allFruits = fruits.concat(moreFruits);
console.log(allFruits); // ["kiwi", "peach", "pear", "orange", "grape", "mango"]

// slice()
let someFruits = allFruits.slice(1, 4);
```

```
console.log(someFruits); // ["peach", "pear", "orange"]

// sort() and reverse()
allFruits.sort();
console.log(allFruits); // ["grape", "kiwi", "mango", "orange", "peach", "pear"]
allFruits.reverse();
console.log(allFruits); // ["pear", "peach", "orange", "mango", "kiwi", "grape"]
```