

# CAP-756

Java Script

By

Dr. Shuja R Mirza

# Getting Started With JavaScript

- JavaScript is a popular programming language that has a wide range of applications.
- Because of its wide range of applications, you can run JavaScript in several ways:
  1. Using console tab of web browsers
  2. Using Node.js
  3. By creating web pages

# Using Console Tab of Web Browsers

- All the popular web browsers have built-in JavaScript engines. Hence, you can run JavaScript on a browser.
- To run JavaScript on a browser
  1. Open your favorite browser (here we will use Google Chrome).
  2. Open the developer tools by right clicking on an empty area and select Inspect. **Shortcut: ctrl+alt+j**
  3. On the developer tools, go to the console tab. Then, write JavaScript code and press enter to run the code.

# Using Node.js

- Node is a back-end run-time environment for executing JavaScript code. To run JS using Node.js, follow these steps:

**1.Install the latest version of [Node.js](#).**

**2.Install an IDE/Text Editor like [Visual Studio Code](#).**

**3.In VS code, create a file > write JS code > save it with .js extension.**

**4.Open up the terminal/command prompt > navigate to the file location > type file name and hit enter.**

**5.You will get output on the terminal.**

Note: It is also possible to run JavaScript on the terminal/command prompt directly. For that, simply type node and press enter. Then you can start writing JS code.

# By Creating Web Pages

- JavaScript was initially created to make web pages interactive, that's why JavaScript and HTML go hand in hand.
- To run JS from a webpage, follow these steps:
  1. **Open VS Code > Go to File > New File > Save it with .html extension.** For example, main.html
  2. Type this doctype (minimum valid HTML code) and save it in the file

```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">

  <title>Programiz</title>
</head>

<body>
  <script src=""></script>
</body>
</html>
```

3. Similarly create a JS file, write the following JS code and save it with **.js** extension like main.js

```
console.log('hello world');
```

4. From inside the HTML file, we need to link the main.js file to use it. You can achieve that by adding the following code in main.html

```
<script scr="main.js"></script>
```

5. Open the main.html using a browser.

# JavaScript Variables and Constants

- Here you will learn about JavaScript variables and constants, and also how to initialize and use them with the help of examples.
- **JavaScript Variables**
  - In programming, a variable is a container (storage area) to hold data. For example,
  - Let num=5;
  - Here num is a variable and is storing 5.
- **JavaScript Declare Variables**
  - In JavaScript, we use either var or let keyword to declare variables. For example, var x; let y;
  - Here x & y are variable

- **JavaScript var Vs let**

- Both **var** and **let** are used to declare variables. However, there are some differences between them.

var	let
<code>var</code> is used in the older versions of JavaScript	<code>let</code> is the new way of declaring variables starting <b>ES6 (ES2015)</b> .
<code>var</code> is function scoped	<code>let</code> is block scoped
For example, <code>var x;</code>	For example, <code>let y;</code>



## JavaScript Initialize Variables

- We use the assignment operator **=** **to** assign a value to a variable.
- **Eg let x;**
- **X=5;**
- You can also initialize variables during its declaration.
- **Let x=5;**
- **Let y=6;**
- In JavaScript, it's possible to declare variables in a single statement.
- **Let x=5, y=7 z=100;**

- If you use a variable without initializing it, it will have an **undefined** value
- `let x; // x is the name of the variable`
- `console.log(x); // undefined`
- Here **x** is the variable name and since it does not contain any value, it will be undefined.
- **Change the Value of Variables**
- It's possible to change the value stored in the variable. For example,
- `// 5 is assigned to variable x`
- `let x = 5;`
- `console.log(x); // 5`
- `// value of variable x is changed`
- `x = 3;`
- `console.log(x); // 3`
- The value of a variable may **vary**. Hence, the name **variable**.

- **JavaScript Constants**
- **The `const` keyword was also introduced in the ES6(ES2015) version to create constants. For example,**
- **`const x = 5;`**
- Once a constant is initialized, we cannot change its value.
- **`const x = 5;`**
- **`x = 10; // Error! constant cannot be changed.`**
- **`console.log(x)`**
- Simply, a constant is a type of variable whose value cannot be changed.
- Also, you cannot declare a constant without initializing it. For example,
- `const x; // Error! Missing initializer in const declaration.`
- `x = 5;`
- `console.log(x)`

- **JavaScript console.log()**
- All modern browsers have a web console for debugging.
- The **console.log()** method is used to write messages to these consoles. For example,
- Let sum=44;
- Console.log(sum);
- When you run the above code, **44** is printed on the console.
- **console.log() Syntax**
- Its syntax is:
- **console.log(message);**
- Here, the message refers to either a variable or a value.

- Example 1: Print a Sentence

// program to print a sentence

// passing string

```
console.log("I love JS");
```

- **Output:**

- I love JS

- Example 2: Print Values Stored in Variables

// program to print variables values

// storing values

```
const greet = 'Hello';
```

```
const name = 'Jack';
```

```
console.log(greet + ' ' + name);
```

- **Output**

- Hello Jack

- **JavaScript Data Types**

- There are different types of data that we can use in a JavaScript program. For example,

- `const x = 5;`

- `const y = "Hello";`

- **JavaScript Data Types**

- There are eight basic data types in JavaScript. They are:

Data Types	Description	Example
<code>String</code>	represents textual data	<code>'hello'</code> , <code>"hello world!"</code> etc
<code>Number</code>	an integer or a floating-point number	<code>3</code> , <code>3.234</code> , <code>3e-2</code> etc.
<code>BigInt</code>	an integer with arbitrary precision	<code>900719925124740999n</code> , <code>1n</code> etc.
<code>Boolean</code>	Any of two values: true or false	<code>true</code> and <code>false</code>
<code>undefined</code>	a data type whose variable is not initialized	<code>let a;</code>
<code>null</code>	denotes a <code>null</code> value	<code>let a = null;</code>
<code>Symbol</code>	data type whose instances are unique and immutable	<code>let value = Symbol('hello');</code>
<code>Object</code>	key-value pairs of collection of data	<code>let student = { };</code>

- Here, all data types except **Object** are primitive data types, whereas Object is non-primitive.
- **Note:** The Object data type (non-primitive type) can store collections of data, whereas primitive data type can only store a single data.
- **JavaScript String**
- **String** is used to store text. In JavaScript, strings are surrounded by quotes:
- Single quotes: 'Hello'
- Double quotes: "Hello"
- Backticks: `Hello`
- For example,
- //strings example
- `const name = 'ram';`
- `const name1 = "hari";`
- `const result = `The names are ${name} and ${name1}`;`
- Single quotes and double quotes are practically the same and you can use either of them.



- **JavaScript Number**

- Number represents integer and floating numbers (decimals and exponentials). For example,
- `const number1 = 3;`
- `const number2 = 3.433;`
- `const number3 = 3e5 // 3 * 10^5`
- A number type can also be **+Infinity**, **-Infinity**, and **NaN** (not a number). For example,
- `const number1 = 3/0;`
- `console.log(number1); // Infinity`
- `const number2 = -3/0;`
- `console.log(number2); // -Infinity`
- `// strings can't be divided by numbers`
- `const number3 = "abc"/3;`
- `console.log(number3); // NaN`

- **JavaScript Boolean**

- This data type represents logical entities. Boolean represents one of two values: **true or false**. It is easier to think of it as a yes/no switch. For example,
- `const dataChecked = true;`
- `const valueCounted = false;`

- **JavaScript undefined**

- The undefined data type represents value that is not assigned. If a variable is declared but the value is not assigned, then the value of that variable will be undefined. For example,
- `let name;`
- `console.log(name); // undefined`
- It is also possible to explicitly assign a variable value undefined. For example,
- `let name = undefined;`
- `console.log(name); // undefined`

- **JavaScript null**

- In JavaScript, null is a special value that represents empty or unknown value. For example,
- `const number = null;`
- The code above suggests that the number variable is empty.

- **JavaScript Symbol**

- This data type was introduced in a newer version of JavaScript (from ES2015).
- A value having the data type **Symbol** can be referred to as a **symbol** value. **Symbol** is an immutable primitive value that is unique. For example,
- **// two symbols with the same description**
- `const value1 = Symbol('hello');`
- `const value2 = Symbol('hello');`
- Though **value1** and **value2** both contain 'hello', they are different as they are of the **Symbol** type.

- **JavaScript Object**

- An object is a complex data type that allows us to store collections of data. For example,

```
const student = {  
  firstName: 'ram',  
  lastName: null,  
  class: 10  
};
```

- **JavaScript Type**

- **JavaScript is a dynamically typed (loosely typed) language. JavaScript automatically determines the variables' data type for you.**

- It also means that a variable can be of one data type and later it can be changed to another data type. For example,

```
// data is of undefined type
```

```
let data;
```

```
// data is of integer type
```

```
data = 5;
```

```
// data is of string type
```

```
data = "JavaScript Programming";
```

- **JavaScript typeof**

- To find the type of a variable, you can use the typeof operator. For example,

- `const name = 'ram';`

- `typeof(name); // returns "string"`

- `const number = 4;`

- `typeof(number); //returns "number"`

- `const valueChecked = true;`

- `typeof(valueChecked); //returns "boolean"`

- `const a = null;`

- `typeof(a); // returns "object"`

- **JavaScript Operators**
- **What is an Operator?**
- In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables). For example,
- **2 + 3; // 5**

Here + is an operator that performs addition, and 2 and 3 are operands.

- **JavaScript Operator Types**

1. Assignment Operators
2. Arithmetic Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. String Operators
7. Other Operators

## • JavaScript Assignment Operators

- Assignment operators are used to **assign** values to variables. For example,
- `Const x=5;`
- Here, the `=` operator is used to assign value 5 to variable x.

Operator	Name	Example
<code>=</code>	Assignment operator	<code>a = 7; // 7</code>
<code>+=</code>	Addition assignment	<code>a += 5; // a = a + 5</code>
<code>-=</code>	Subtraction Assignment	<code>a -= 2; // a = a - 2</code>
<code>*=</code>	Multiplication Assignment	<code>a *= 3; // a = a * 3</code>
<code>/=</code>	Division Assignment	<code>a /= 2; // a = a / 2</code>
<code>%=</code>	Remainder Assignment	<code>a %= 2; // a = a % 2</code>
<code>**=</code>	Exponentiation Assignment	<code>a **= 2; // a = a**2</code>

- **JavaScript Arithmetic Operators**
- Arithmetic operators are used to perform **arithmetic calculations**. For example, `const number = 3 + 5; // 8`
- **Here, the + operator is used to add two operands.**

Operator	Name	Example
<code>+</code>	Addition	<code>x + y</code>
<code>-</code>	Subtraction	<code>x - y</code>
<code>*</code>	Multiplication	<code>x * y</code>
<code>/</code>	Division	<code>x / y</code>
<code>%</code>	Remainder	<code>x % y</code>
<code>++</code>	Increment (increments by 1)	<code>++x</code> or <code>x++</code>
<code>--</code>	Decrement (decrements by 1)	<code>--x</code> or <code>x--</code>
<code>**</code>	Exponentiation (Power)	<code>x ** y</code>



```
let x = 5;
let y = 3;

// addition
console.log('x + y = ', x + y); // 8

// subtraction
console.log('x - y = ', x - y); // 2

// multiplication
console.log('x * y = ', x * y); // 15

// division
console.log('x / y = ', x / y); // 1.6666666666666667

// remainder
console.log('x % y = ', x % y); // 2

// increment
console.log('++x = ', ++x); // x is now 6
console.log('x++ = ', x++); // prints 6 and then increased to 7
console.log('x = ', x); // 7

// decrement
console.log('--x = ', --x); // x is now 6
console.log('x-- = ', x--); // prints 6 and then decreased to 5
console.log('x = ', x); // 5

//exponentiation
console.log('x ** y =', x ** y);
```

- **JavaScript Comparison Operators**

- Comparison operators compare two values and return a boolean value, either true or false. For example,

- `const a = 3, b = 2;`

- `console.log(a > b); // true`

- Here, the comparison operator `>` is used to compare whether a is greater than b.

Operator	Description	Example
<code>==</code>	<b>Equal to:</b> returns <code>true</code> if the operands are equal	<code>x == y</code>
<code>!=</code>	<b>Not equal to:</b> returns <code>true</code> if the operands are not equal	<code>x != y</code>
<code>===</code>	<b>Strict equal to:</b> <code>true</code> if the operands are equal and of the same type	<code>x === y</code>
<code>!==</code>	<b>Strict not equal to:</b> <code>true</code> if the operands are equal but of different type or not equal at all	<code>x !== y</code>
<code>&gt;</code>	<b>Greater than:</b> <code>true</code> if left operand is greater than the right operand	<code>x &gt; y</code>
<code>&gt;=</code>	<b>Greater than or equal to:</b> <code>true</code> if left operand is greater than or equal to the right operand	<code>x &gt;= y</code>
<code>&lt;</code>	<b>Less than:</b> <code>true</code> if the left operand is less than the right operand	<code>x &lt; y</code>
<code>&lt;=</code>	<b>Less than or equal to:</b> <code>true</code> if the left operand is less than or equal to the right operand	<code>x &lt;= y</code>

- **Example 2: Comparison operators in JavaScript**

```
// equal operator
console.log(2 == 2); // true
console.log(2 == '2'); // true

// not equal operator
console.log(3 != 2); // true
console.log('hello' != 'Hello'); // true

// strict equal operator
console.log(2 === 2); // true
console.log(2 === '2'); // false

// strict not equal operator
console.log(2 !== '2'); // true
console.log(2 !== 2); // false
```

## • JavaScript Logical Operators

- Logical operators perform logical operations and return a boolean value, either true or false. For example,
- `const x = 5, y = 3;`
- `(x < 6) && (y < 5); // true`
- Here, `&&` is the logical operator AND. Since both `x < 6` and `y < 5` are true, the result is true.

Operator	Description	Example
<code>&amp;&amp;</code>	<b>Logical AND:</b> <code>true</code> if both the operands are <code>true</code> , else returns <code>false</code>	<code>x &amp;&amp; y</code>
<code>  </code>	<b>Logical OR:</b> <code>true</code> if either of the operands is <code>true</code> ; returns <code>false</code> if both are <code>false</code>	<code>x    y</code>
<code>!</code>	<b>Logical NOT:</b> <code>true</code> if the operand is <code>false</code> and vice-versa.	<code>!x</code>

## Example 3: Logical Operators in JavaScript



```
// logical AND
console.log(true && true); // true
console.log(true && false); // false

// logical OR
console.log(true || false); // true

// logical NOT
console.log(!true); // false
```

- **JavaScript Bitwise Operators**

- Bitwise operators perform operations on binary representations of numbers.

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Sign-propagating right shift
>>>	Zero-fill right shift

- JavaScript String Operators
- In JavaScript, you can also use the + operator to concatenate (join) two or more strings.
- **Example 4: String operators in JavaScript**

```
// concatenation operator
console.log('hello' + 'world');

let a = 'JavaScript';

a += ' tutorial'; // a = a + ' tutorial';
console.log(a);
```

## Output

```
helloworld
JavaScript tutorial
```



## • Other JavaScript Operators

Operator	Description	Example
<code>,</code>	evaluates multiple operands and returns the value of the last operand.	<code>let a = (1, 3 , 4); // 4</code>
<code>?:</code>	returns value based on the condition	<code>(5 &gt; 3) ? 'success' : 'error'; // "success"</code>
<code>delete</code>	deletes an object's property, or an element of an array	<code>delete x</code>
<code>typeof</code>	returns a string indicating the data type	<code>typeof 3; // "number"</code>
<code>void</code>	discards the expression's return value	<code>void(x)</code>
<code>in</code>	returns <code>true</code> if the specified property is in the object	<code>prop in object</code>
<code>instanceof</code>	returns <code>true</code> if the specified object is of of the specified object type	<code>object instanceof object_type</code>

# JavaScript Comparison and Logical Operators

- Comparison operators compare two values and give back a Boolean value: either true or false. Comparison operators are used in decision making and loops.

Operator	Description	Example
<code>==</code>	<b>Equal to:</b> <code>true</code> if the operands are equal	<code>5==5; //true</code>
<code>!=</code>	<b>Not equal to:</b> <code>true</code> if the operands are not equal	<code>5!=5; //false</code>
<code>===</code>	<b>Strict equal to:</b> <code>true</code> if the operands are equal and of the same type	<code>5=== '5'; //false</code>
<code>!==</code>	<b>Strict not equal to:</b> <code>true</code> if the operands are equal but of different type or not equal at all	<code>5!== '5'; //true</code>
<code>&gt;</code>	<b>Greater than:</b> <code>true</code> if the left operand is greater than the right operand	<code>3&gt;2; //true</code>
<code>&gt;=</code>	<b>Greater than or equal to:</b> <code>true</code> if the left operand is greater than or equal to the right operand	<code>3&gt;=3; //true</code>
<code>&lt;</code>	<b>Less than:</b> <code>true</code> if the left operand is less than the right operand	<code>3&lt;2; //false</code>
<code>&lt;=</code>	<b>Less than or equal to:</b> <code>true</code> if the left operand is less than or equal to the right operand	<code>2&lt;=2; //true</code>

- **Example 1: Equal to Operator**

- `const a = 5, b = 2, c = 'hello';`
- `// equal to operator`
- `console.log(a == 5); // true`
- `console.log(b == '2'); // true`
- `console.log(c == 'Hello'); // false`

- **Example 2: Not Equal to Operator**

- `const a = 3, b = 'hello';`
- `// not equal operator`
- `console.log(a != 2); // true`
- `console.log(b != 'Hello'); // true`

- **Example 3: Strict Equal to Operator**

- `const a = 2;`
- `// strict equal operator`
- `console.log(a === 2); // true`
- `console.log(a === '2'); // false`

- **Example 4: Strict Not Equal to Operator**

- `const a = 2, b = 'hello';`
- `// strict not equal operator`
- `console.log(a !== 2); // false`
- `console.log(a !== '2'); // true`
- `console.log(b !== 'Hello'); // true`

- **Example 5: Greater than Operator**

- `const a = 3;`
- `// greater than operator`
- `console.log(a > 2); // true`

- **Example 6: Greater than or Equal to Operator**

- `const a = 3;`
- `// greater than or equal operator`
- `console.log(a >= 3); //true`

- **Example 7: Less than Operator**

- `const a = 3, b = 2;`
- `// less than operator`
- `console.log(a < 2); // false`
- `console.log(b < 3); // true`

- **Example 8: Less than or Equal to Operator**

- `const a = 2;`
- `// less than or equal operator`
- `console.log(a <= 3) // true`
- `console.log(a <= 2); // true`

- **JavaScript Logical Operators**

- Logical operators perform logical operations: **AND**, **OR** and **NOT**.

Operator	Description	Example
&&	<b>Logical AND:</b> true if both the operands/boolean values are true, else evaluates to false	true && false; // false
	<b>Logical OR:</b> true if either of the operands/boolean values is true. evaluates to false if both are false	true    false; // true
!	<b>Logical NOT:</b> true if the operand is false and vice-versa.	!true; // false

- **Example 9: Logical AND Operator**

- `const a = true, b = false;`
- `const c = 4;`
- `// logical AND`
- `console.log(a && a); // true`
- `console.log(a && b); // false`
- `console.log((c > 2) && (c < 2)); // false`

- **Example 10: Logical OR Operator**

- `const a = true, b = false, c = 4;`
- `// logical OR`
- `console.log(a || b); // true`
- `console.log(b || b); // false`
- `console.log((c > 2) || (c < 2)); // true`

- **Example 11: Logical NOT Operator**

- `const a = true, b = false;`
- `// logical NOT`
- `console.log(!a); // false`
- `console.log(!b); // true`

## JavaScript if...else Statement

- In computer programming, there may arise situations where you have to run a block of code among more than one alternatives.
- For example, assigning grades **A**, **B** or **C** based on marks obtained by a student.
- In such situations, you can use the JavaScript if...else statement to create a program that can make decisions.
- **In JavaScript, there are three forms of the if...else statement.**
  - 1. if statement**
  - 2. if...else statement**
  - 3. if...else if...else statement**

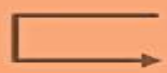


# JavaScript if Statement

- The syntax of the **if** statement is:
- `if (condition) {`
- `// the body of if`
- `}`
- The if statement evaluates the condition inside the parenthesis ().
- If the condition is evaluated to true, the code inside the body of if is executed.
- If the condition is evaluated to false, the code inside the body of if is skipped.


## Condition is true

```
let number = 2;  
if (number > 0) {  
    // code  
}  
  
//code after if
```

A flowchart showing a horizontal arrow pointing to the opening curly brace of the if block, then a vertical arrow pointing down to the closing curly brace, and finally a horizontal arrow pointing to the code after the if block.

## Condition is false

```
let number = -2;  
if (number > 0) {  
    // code  
}  
  
//code after if
```


A flowchart showing a horizontal arrow pointing to the opening curly brace of the if block, then a vertical arrow pointing down and then a horizontal arrow pointing to the code after the if block, bypassing the body of the if block.

# JavaScript if...else statement

- An if statement can have an optional else clause. The syntax of the if...else statement is:
- `if (condition) {`
- `// block of code if condition is true`
- `} else {`
- `// block of code if condition is false`
- `}`

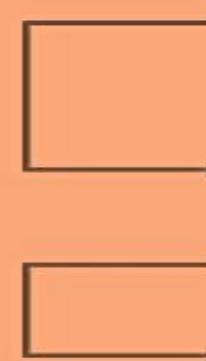
## Condition is true

```
let number = 2;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if
```



## Condition is false

```
let number = -2;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if
```



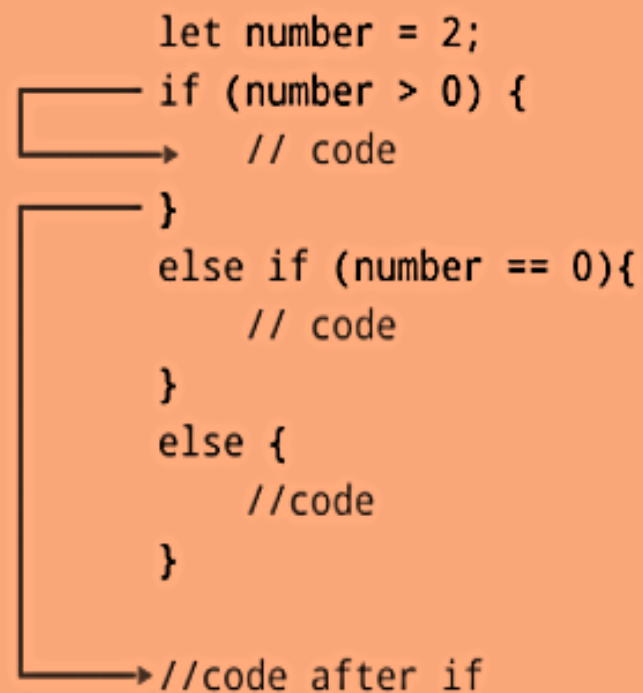
Working of the if...else statement

- **JavaScript if...else if statement**

- The if...else statement is used to execute a block of code among two alternatives. However, if you need to make a choice between more than two alternatives, if...else if...else can be used.
- The syntax of the if...else if...else statement is:
  - if (condition1) {
    - // code block 1
  - } else if (condition2){
    - // code block 2
  - } else {
    - // code block 3
  - }

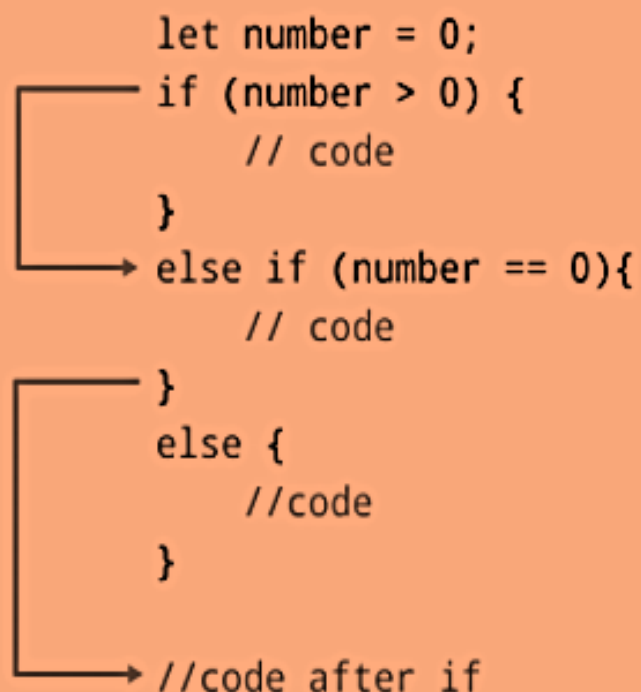
### 1st Condition is true

```
let number = 2;  
if (number > 0) {  
  // code  
}  
else if (number == 0){  
  // code  
}  
else {  
  //code  
}  
//code after if
```

A flowchart illustrating the execution of the code when the first condition is true. It starts with the variable 'number' assigned the value 2. An arrow points to the 'if (number > 0)' condition, which is true. The flow enters the first code block, then exits it and goes down to the 'else if' block, then down to the 'else' block, and finally exits the 'else' block to the code after the if statement.

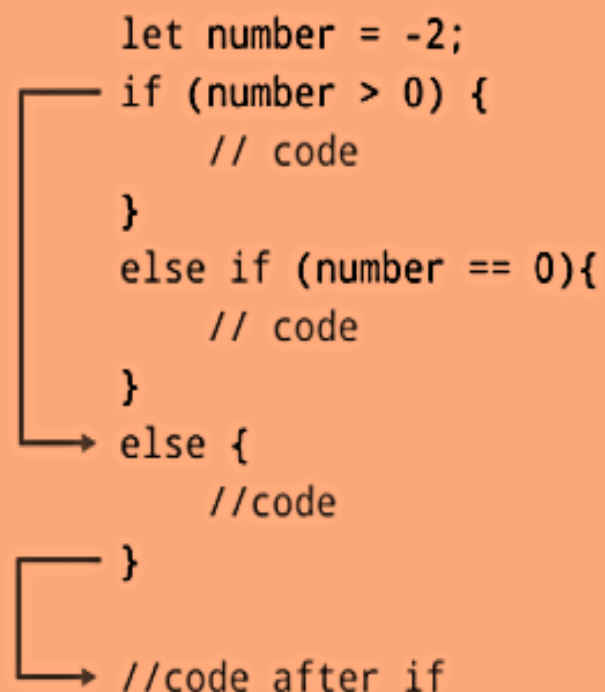
### 2nd Condition is true

```
let number = 0;  
if (number > 0) {  
  // code  
}  
else if (number == 0){  
  // code  
}  
else {  
  //code  
}  
//code after if
```

A flowchart illustrating the execution of the code when the second condition is true. It starts with the variable 'number' assigned the value 0. An arrow points to the 'if (number > 0)' condition, which is false. The flow then goes down to the 'else if (number == 0)' condition, which is true. The flow enters the second code block, then exits it and goes down to the 'else' block, and finally exits the 'else' block to the code after the if statement.

### All Conditions are false

```
let number = -2;  
if (number > 0) {  
  // code  
}  
else if (number == 0){  
  // code  
}  
else {  
  //code  
}  
//code after if
```

A flowchart illustrating the execution of the code when all conditions are false. It starts with the variable 'number' assigned the value -2. An arrow points to the 'if (number > 0)' condition, which is false. The flow then goes down to the 'else if (number == 0)' condition, which is also false. The flow then goes down to the 'else' block, enters it, and finally exits the 'else' block to the code after the if statement.

Working of the if...else if...else statement