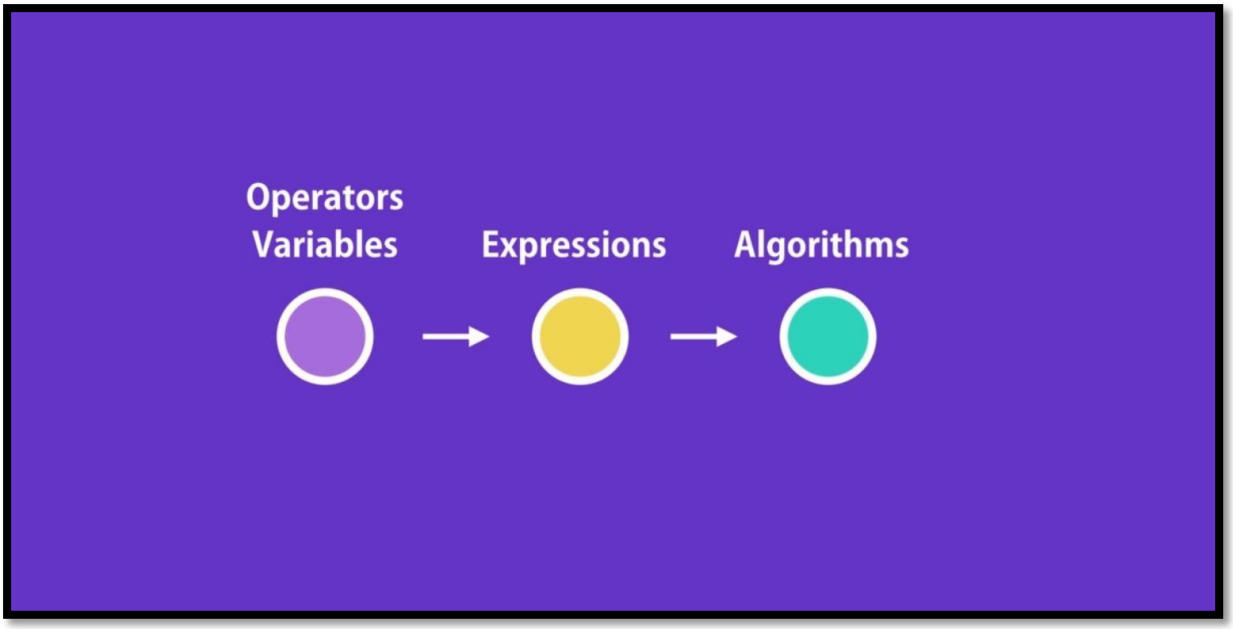
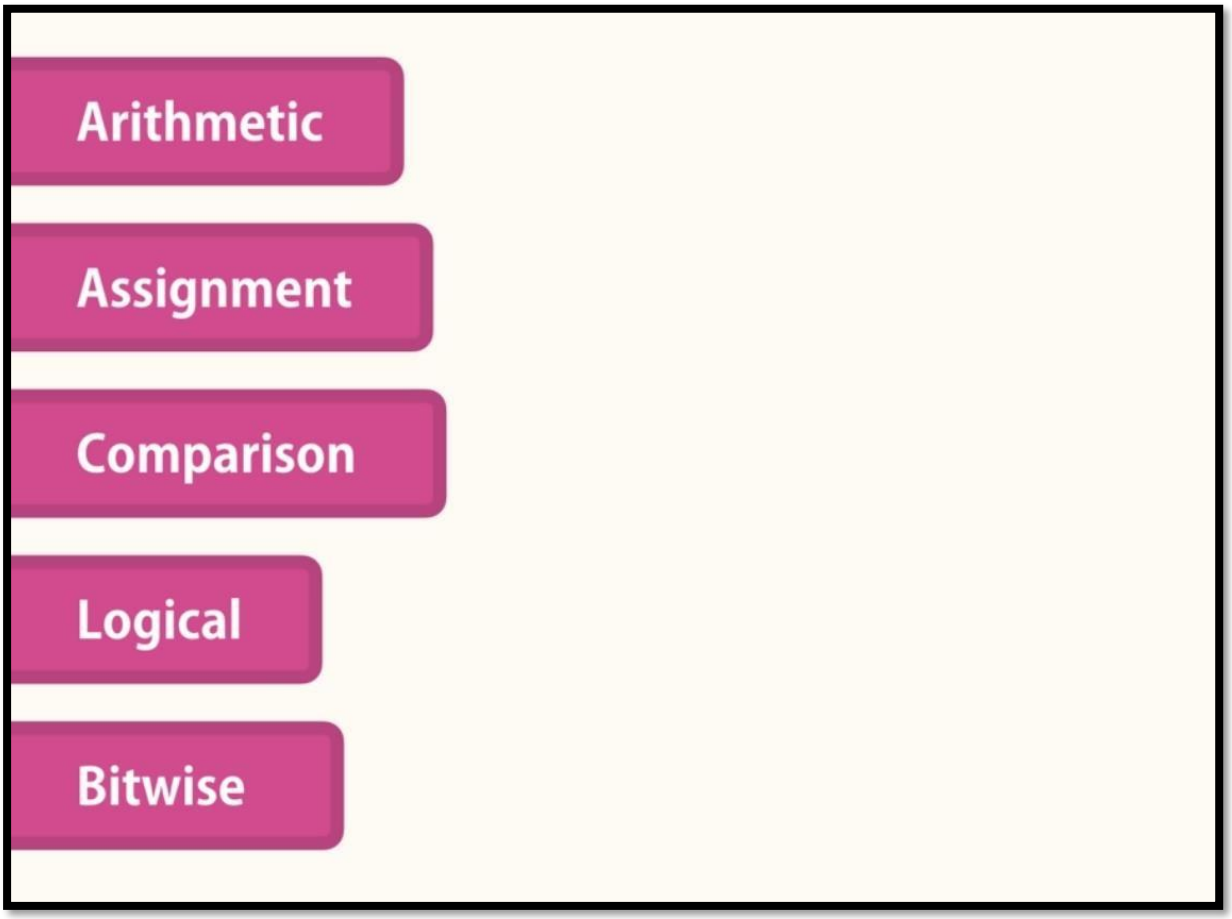
Operators

In Javascript, we have different kind of operators. We use these operators along with variables and constants to create expressions so that we could create logics or implement algorithms.



In Javascript we have following operators



Arithmetic Operators

```
let x = 10;
let y = 50;

console.log(x + y); // Addition
console.log(x - y); // Subtraction
console.log(x * y); // Multiplication
console.log(x / y); // Division
console.log(x % y); // Modulo
console.log(x ** 3); // Exponential
console.log(++x); // Unary Pre-Increment
console.log(x++); // Unary Post-Increment
console.log(--x); // Unary Pre-Decrement
console.log(x--); // Unary Post-Increment
```

Assignment Operators

```
x += y; // shorthand for x = x + y;
x -= y; // shorthand for x = x - y;
x *= y; // shorthand for x = x * y;
x /= y; // shorthand for x = x / y;
x %= y; // shorthand for x = x % y;
x **= 3; // shorthand for x = x ** 3;
```

Comparison Operators

1. >
2. >=
3. <
4. <=

Truthy and Falsy Values

All values have inherent truthy or falsy boolean values.

Falsy Values

1. false
2. 0
3. Empty String ''
4. Null
5. Undefined
6. NaN

Truthy Values

Everything else is truthy.

Equality Operators

1. ==
2. !=
3. ===
4. !==

Strict Equality Operator

===

Lose Equality Operator

==

Strict Equality ensures both the elements are of same type and value while ensures Lose equality compares the truthy and falsy values only.

Ternary Operator

(condition) ? (true statement) : (false statement)

Logical Operators

1. Logical OR ||
2. Logical AND &&
3. Logical NOT !

Logical And

It returns the last truthy value. The moment it sees a falsy value. It immediately returns that as output.

Logical OR

The logical OR operator returns the first truthy value it encounters or the last value if none of the values are truthy.

Logical Operators with Booleans

```
console.log(true && true); // true
console.log(false && true); // false
console.log(false || true); // true
console.log(!false); // true
console.log(!true); // false
```

Logical Operators with Non Booleans

```
console.log(false || 'Mosh'); // Mosh
console.log(false || 1); // 1
console.log(false || 1 || 2); // 1
```

It returns the first truthy value.

Short Circuiting

```
let userColor  =  'red';
let defaultColor = 'blue';
let currentColor = userColor || defaultColor;
console.log(currentColor); //red

let userColor;
let defaultColor = 'blue';
let currentColor = userColor || defaultColor;
console.log(currentColor); //blue
```

Bitwise Operators

1. Bitwise AND
2. Bitwise OR

Control Flow

1. if else
2. switch case
3. for
4. while
5. do while
6. for in
7. for of // ECMAScript ES6
8. break
9. continue

```
const person = { name : 'Anubhav', age: 15 };
const colors = ['red', 'green', 'blue'];

for (let key in person)
{
    console.log(key, person[key]);
}

//name Anubhav
//age 15

for(let idx in colors)
{
    console.log(idx, colors[idx]);
}

//0 red
//1 green
//2 blue

for(let color of colors)
{
    console.log(color);
}

//red
//green
//blue
```

typeof

The typeof operator to find the data type of a JavaScript variable.

```
var  v;
v = "1";
console.log(typeof v); //string
v = 2;
console.log(typeof v); //number
v = true;
console.log(typeof v); //boolean
v = {};
console.log(typeof v); //object
v = Symbol();
console.log(typeof v); //Symbol

let doesNotExist;
console.log(typeof doesNotExist); //undefined
v = null;
console.log(typeof v); //object
```

```
v = function(){};
console.log(typeof v); //function
v = [1, 2, 3];
console.log(typeof v); //object
```

Nullish coalescing operator (??)

The nullish coalescing (??) operator is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand.

```
const foo = null ?? 'default string';
console.log(foo);
// Expected output: "default string"

const baz = 0 ?? 42;
console.log(baz);
// Expected output: 0
```

Optional chaining (?.)

The **optional chaining (?.)** operator accesses an object's property or calls a function. If the object accessed or function called using this operator is undefined or null, the expression short circuits and evaluates to undefined instead of throwing an error.

```
const adventurer =
{
    name: 'Alice',
    cat:
    {
      name: 'Dinah'
    }
};

const dogName = adventurer.dog?.name;
console.log(dogName);
// Expected output: undefined

console.log(adventurer.someNonExistentMethod?.());
// Expected output: undefined
```


Optional chaining operator
obj.address?.village
If "address" property is available in the "obj" object then we will able to access the "village" nested property.
If "address" property is not null/undefined then "village" property will be evaluated, otherwise we will get "undefined" as output.


LONG HAND
const App = (props) =>{
<>
<h1>{props && prop.name}</h1>
{props && props.age}
</>
}
SHORT HAND
const App = ({name, age}) =>{
<>
<h1>{props?.name}</h1>
{props?.age}
</>
}