Here’s the list with one-line descriptions for all JavaScript operators

### 1. \*\*Arithmetic Operators\*\*

- `+` : Addition — Adds two numbers or concatenates strings.

- `-` : Subtraction — Subtracts the right operand from the left.

- `\*` : Multiplication — Multiplies two numbers.

- `/` : Division — Divides the left operand by the right.

- `%` : Modulus — Returns the remainder of division.

- `++` : Increment — Increases a variable's value by 1.

- `--` : Decrement — Decreases a variable's value by 1.

- `\*\*` : Exponentiation — Raises the left operand to the power of the right.

### 2. \*\*Assignment Operators\*\*

- `=` : Assignment — Assigns the right-hand value to the left-hand variable.

- `+=` : Addition assignment — Adds and assigns the result to the variable.

- `-=` : Subtraction assignment — Subtracts and assigns the result to the variable.

- `\*=` : Multiplication assignment — Multiplies and assigns the result to the variable.

- `/=` : Division assignment — Divides and assigns the result to the variable.

- `%=` : Remainder assignment — Assigns the remainder of division to the variable.

- `\*\*=` : Exponentiation assignment — Assigns the result of raising the variable to a power.

- `<<=` : Left shift assignment — Shifts bits to the left and assigns the result.

- `>>=` : Right shift assignment — Shifts bits to the right and assigns the result.

- `>>>=` : Unsigned right shift assignment — Shifts bits right, filling with zeros, and assigns the result.

- `&=` : Bitwise AND assignment — Performs bitwise AND and assigns the result.

- `^=` : Bitwise XOR assignment — Performs bitwise XOR and assigns the result.

- `|=` : Bitwise OR assignment — Performs bitwise OR and assigns the result.

- `??=` : Nullish coalescing assignment — Assigns the right-hand value if the left-hand side is `null` or `undefined`.

### 3. \*\*Comparison Operators\*\*

- `==` : Equal to — Compares two values for equality after type coercion.

- `!=` : Not equal to — Compares two values for inequality after type coercion.

- `===` : Strict equal to — Compares two values for equality without type conversion.

- `!==` : Strict not equal to — Compares two values for inequality without type conversion.

- `>` : Greater than — Checks if the left operand is greater than the right.

- `<` : Less than — Checks if the left operand is less than the right.

- `>=` : Greater than or equal to — Checks if the left operand is greater than or equal to the right.

- `<=` : Less than or equal to — Checks if the left operand is less than or equal to the right.

### 4. \*\*Logical Operators\*\*

- `&&` : Logical AND — Returns true if both operands are true.

- `||` : Logical OR — Returns true if either operand is true.

- `!` : Logical NOT — Inverts the boolean value of the operand.

- `??` : Nullish coalescing — Returns the right operand if the left is `null` or `undefined`.

### 5. \*\*Bitwise Operators\*\*

- `&` : Bitwise AND — Performs a bitwise AND between two operands.

- `|` : Bitwise OR — Performs a bitwise OR between two operands.

- `^` : Bitwise XOR — Performs a bitwise XOR between two operands.

- `~` : Bitwise NOT — Inverts the bits of its operand.

- `<<` : Left shift — Shifts the bits of the left operand to the left by the number of positions specified by the right operand.

- `>>` : Right shift — Shifts the bits of the left operand to the right by the number of positions specified by the right operand.

- `>>>` : Unsigned right shift — Shifts the bits to the right, filling with zeros, regardless of the sign.

### 6. \*\*Ternary Operator\*\*

- `? :` : Conditional operator — Returns one of two values based on a condition (`condition ? valueIfTrue : valueIfFalse`).

### 7. \*\*Type Operators\*\*

- `typeof` : Type operator — Returns the type of a variable (e.g., `string`, `number`).

- `instanceof` : Instance operator — Checks if an object is an instance of a constructor.

- `in` : Property existence — Checks if a property exists in an object or array.

- `void` : Void operator — Evaluates an expression and returns `undefined`.

- `delete` : Delete operator — Removes a property from an object.

### 8. \*\*Spread and Rest Operators\*\*

- `...` : Spread — Expands an iterable (like an array) or object properties.

- `...` : Rest — Collects multiple elements into a single array or object (in function arguments or destructuring).

### 9. \*\*Comma Operator\*\*

- `,` : Comma operator — Evaluates multiple expressions and returns the value of the last one.

### 10. \*\*Optional Chaining Operators\*\*

- `?.` : Optional chaining — Accesses a property or method only if the object is not `null` or `undefined`.

- `?.()` : Optional chaining with function calls — Calls a function only if it exists.

- `?.[]` : Optional chaining with arrays or objects — Accesses an array element or object property only if the object exists.

### 11. \*\*Control Flow Operators\*\*

- `await` : Await operator — Pauses an `async` function until a promise is resolved.

- `new` : New operator — Creates a new instance of an object.

- `this` : This operator — Refers to the current object or execution context.

- `super` : Super operator — Calls the parent class constructor (used in subclasses).

- `yield` : Yield operator — Pauses and resumes a generator function.

- `yield\*` : Delegates generator function control to another generator.

### 12. \*\*Module Operators\*\*

- `import` : Import operator — Imports a module or part of a module.

- `export` : Export operator — Exports a module or part of a module.

### 13. \*\*Grouping Operator\*\*

- `()` : Grouping operator — Controls the precedence of evaluation in expressions.