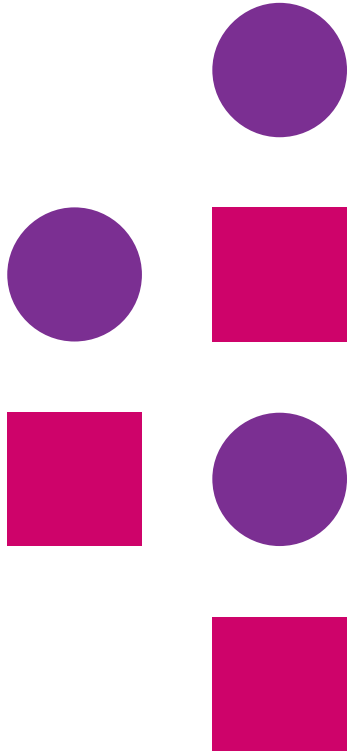
WSA | Forward looking IT finishing school

Operators

JavaScript

Table of Content

- Operators



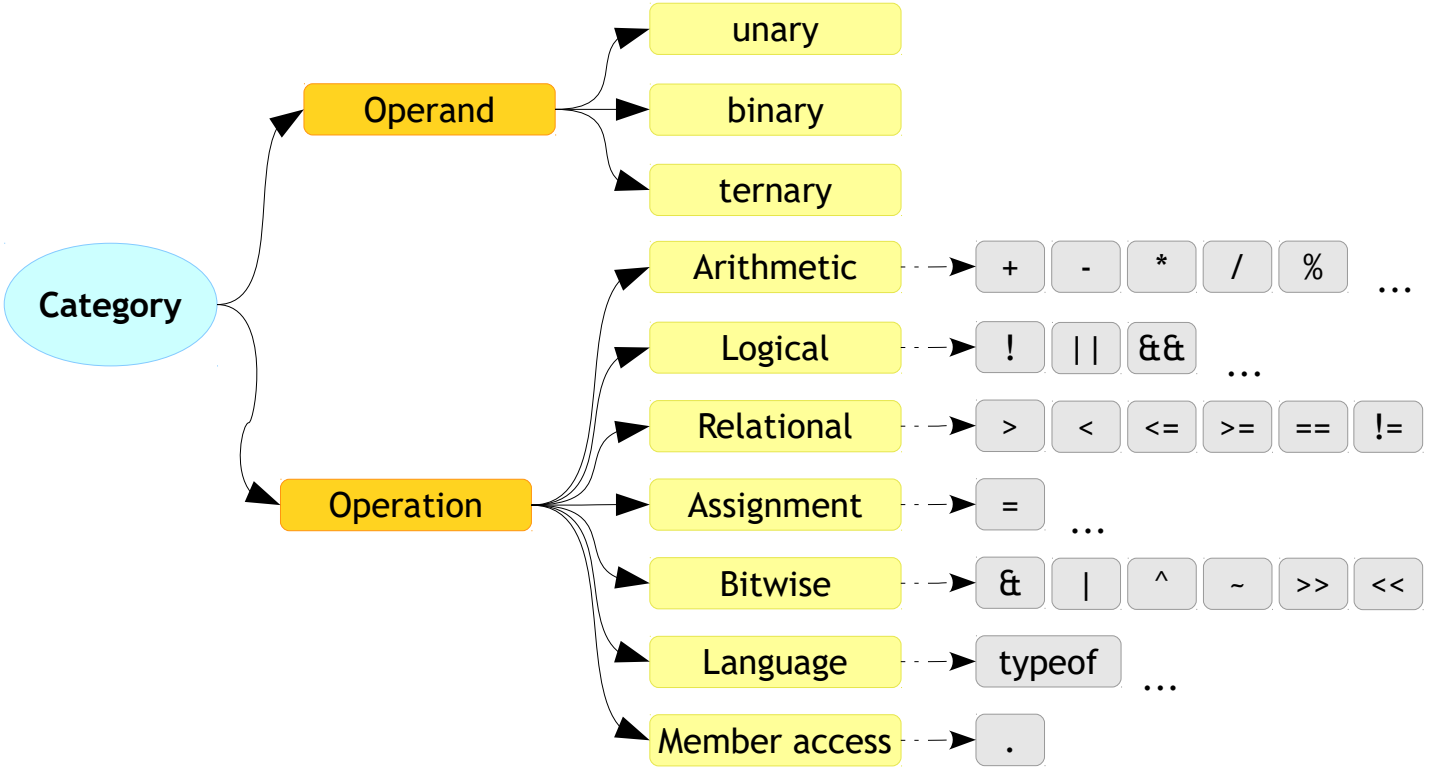
Operators

(JavaScript)

JS - Operators

- Symbols that instructs the compiler to perform specific arithmetic or logical operation on operands
- All operators prominently do two things
  - Operate on its operands
  - Return a result

JS - Operators



JS - Operators

- JavaScript Supports following operators:
  - Arithmetic Operators
  - Assignment Operators
  - Comparison Operators
  - Ternary/Conditional Operator
  - Logical Operators
  - Type Operators
  - Bitwise Operators

JS – Operators Table

Operator	Associativity
( )	NA
. [ ]	left-to-right
new (w/ argument list)	NA
function call	left-to-right
new (w/o argument list)	right-to-left
exp++ exp--	right-to-left
! ~ ++exp --exp + - void typeof delete	right-to-left
* / %	left-to-right
+ -	left-to-right
<< >> >>>	left-to-right

JS – Operators Table

Operator	Associativity
< <= > >= in instanceof	left-to-right
== != === !==	left-to-right
&	left-to-right
^	left-to-right
|	left-to-right
&&	left-to-right
||	left-to-right
? :	right-to-left
yield	right-to-left
= += -= *= /= %= <<= >>= >>>== &= |= ^=	right-to-left
,	left-to-right

Arithmetic operator

Name	Description
+	Returns arithmetic addition
-	Return arithmetic subtraction
*	Returns arithmetic multiplication
/	Returns arithmetic division
%	Returns modulus (remainder)

Unary operators

Name	Description
+	Converts its operand to Number type
-	Converts its operand to Number type and then negates it
++exp	Increment the value by one and store back in variable. Returns new incremented value
--exp	Decrement the value by one and store back in variable. Returns new decremented value
exp++	Returns the old value. Increment the value by one and store back in variable.
exp--	Returns the old value. Decrement the value by one and store back in variable.

Assignment operator

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
>>>=	a >>>= b	a = a >>> b
&=	a &= b	a = a & b
^=	a ^= b	a = a ^ b
|=	a |= b	a = a |= b

Comparison operator

Operator	Description
==	Equal to (compare values)
===	Equal value and equal type
!=	Not equal to
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Comparison operator

- When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison
- An empty string converts to 0
- A non-numeric string converts to NaN which is always false
- Two strings are compared in alphabetical order
- Objects can’t be compared

Ternary/Conditional

- Used as a shortcut for standard “if” statement
- It takes three operands

Syntax:

Condition ? expr1 : expr2

Example:

```
status = (marks >= 50) ? "Pass" : "Fail";
```

Class Work - Ternary

- WAP to find the max of given two numbers
- WAP to return the absolute value of a number
- WAP to check if given number is even or odd

Logical operators

```
<script>
        var num1 = 1, num2 = 0;

        if (++num1 || num2++) {
            document.write("Wow! its true!!");
        }
        else {
            document.write("Hmm! its false");
        }

        document.write("<br> num1 = " + num1);
        document.write("<br> num2 = " + num2);
</script>
```

Name	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical operators

- Logical expressions are evaluated left to right
- They are tested for possible "short-circuit" evaluation using the following rules
  - false && (don’t care exp) is short-circuit evaluated to false
  - true || (don’t care exp) is short-circuit evaluated to true
- Please note that “don’t care exp” is not evaluated

Boolean conversion to false

The Boolean value of 0 (zero) is false	Boolean(0)
The Boolean value of -0 (minus zero) is false	Boolean(-0)
The Boolean value of "" (empty string) is false	Boolean(“”)
The Boolean value of undefined is false	Boolean(undefined)
The Boolean value of null is false	Boolean(null)
The Boolean value of NaN is false	Boolean(NaN)

The typeof operator

- The typeof operator is used to get the data type of its operand
- The operand can be either a literal or a data structure such as a variable, a function, or an object
- The operator returns the data type

```
Syntax:
typeof operand or typeof (operand)
```

The typeof operator

- There are six possible values that typeof returns: object, boolean, function, number, string and undefined

```
Example :
typeof “abc” // returns string
typeof Nan   // returns number
typeof false    //returns boolean
typeof [3,4,5,6] //returns Object
```

The instanceof operator

- The instanceof operator returns true if the specified object is an instance of the specified object

Syntax :

```
var result = <objectName> instanceof <objectType>;
```

The delete operator

- The delete operator removes an object's property completely
- Delete operator removes an element from array; array length does not get affected
- The operator returns true on successful deletion, else false will be returned

The delete operator

Syntax :

```
delete objectName.property;
delete objectName['property'];
delete arrayName[index];
```

Example :

```
delete arr[0];
```

The delete operator

- Delete operator has nothing to do with directly freeing memory (does not free memory)
- If the property which is being deleted does not exist, delete will not have any effect and will return true
- If a property with the same name exists on the object's prototype chain, then, after deletion, the object will use the property from the prototype chain (in other words, delete only has an effect on own properties)

The delete operator

- Non-configurable properties cannot be removed
  - This includes properties of built-in objects like Math, Array, Object and properties that are created as non-configurable with methods like Object.defineProperty()
  - var, let and const create non-configurable properties that cannot be deleted

The new operator

- The new operator is used to create an instance of a user-defined object type or one of built in object types which have a constructor function

Syntax:

```
var objectName = new objectType(param1, param2, ...., paramN);
```

The comma operator

- The comma operator (,) is used to execute two expressions sequentially
- The value of right operand is used as result of comma operator

Syntax :

left-operand, right-operand;

The this operator

- The this operator is used to refer to current object

Syntax:

this.propertyName

The void operator

- The void operator is used to evaluate a JavaScript expression without returning a value

Syntax :

void (expression)

void expression

The void operator

- This operator allows evaluating expressions that produce a value into places where an expression that evaluates to undefined is desired
- The void operator is often used merely to obtain the undefined primitive value, usually using "void(0)"

Example :

```
num1 = void(0);
```

Bitwise operator

- All numbers are stored as 64 bits floating point in JavaScript
- All bitwise operations are performed on 32 bits binary numbers
- Therefore, 64 bits floating point numbers are converted to 32 bits integers by JavaScript before performing bitwise operation
- The result of bitwise operation is a signed 32-bit integer
- The result of bitwise operation is converted back to 64 bits JavaScript number

Primitive data type (Number)

- Integers are like whole numbers, but allow negative numbers and no fraction
- An example of 13_{10} in 32 bit system would be

Bit	No of Bits																													
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

Primitive data type (Number)

- Negative Integers represented with the 2's complement of the positive number
- An example of -13_{10} in 32 bit system would be

Bit	No of Bits																													
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1's Compli	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0
Add 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2's Compli	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1

- Mathematically : $-k \equiv 2^n - k$

Bitwise operator

Operator	Name	Description
&	Bitwise AND	Performs bitwise AND operation
|	Bitwise OR	Performs bitwise OR operation
~	Bitwise NOT	Also known as complement; It flips the bits
^	Bitwise XOR	Performs bitwise XOR operation
<<	Left shift (zero-fill)	Performs bitwise left shift zero filling operation
>>	Right shift (sign-fill)	Performs bitwise right shift sign filling operation
>>>	Right shift (zero-fill)	Performs bitwise right shift zero filling operation

Bitwise operators

&	**Bitwise AND**	Bitwise ANDing of all the bits in two operands
|	**Bitwise OR**	Bitwise ORing of all the bits in two operands

Operand	Value	
A	0x61	0 1 1 0 0 0 0 1
B	0x13	0 0 0 1 0 0 1 1
A & B	0x01	0 0 0 0 0 0 0 1

Operand	Value	
A	0x61	0 1 1 0 0 0 0 1
B	0x13	0 0 0 1 0 0 1 1
A | B	0x73	0 1 1 1 0 0 1 1

Bitwise operators

^	**Bitwise XOR**	Bitwise XORing of all the bits in two operands

Operand	Value	
A	0x61	0 1 1 0 0 0 0 1
B	0x13	0 0 0 1 0 0 1 1
A ^ B	0x72	0 1 1 1 0 0 1 0

~	**Complement**	Complementing all the bits of the operand

Operand	Value	
A	0x61	0 1 1 0 0 0 0 1
~A	0x9E	1 0 0 1 1 1 1 0

Bitwise operators

Left Shift :

shift-expression << additive-expression

(left operand) (right operand)

Right Shift :

shift-expression >> additive-expression

(left operand) (right operand)

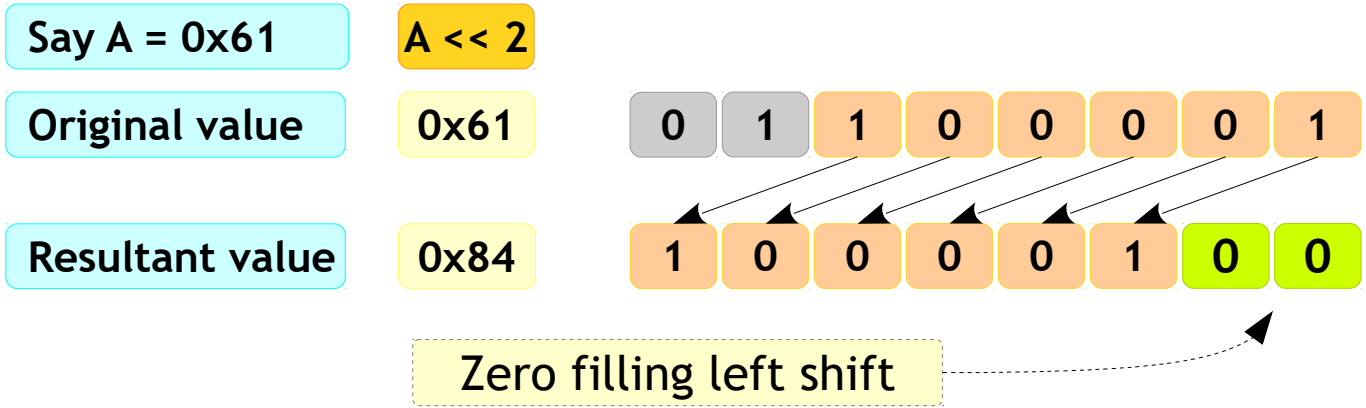
shift-expression >>> additive-expression

(left operand) (right operand)

Bitwise operators

'Value' << 'Bits Count'

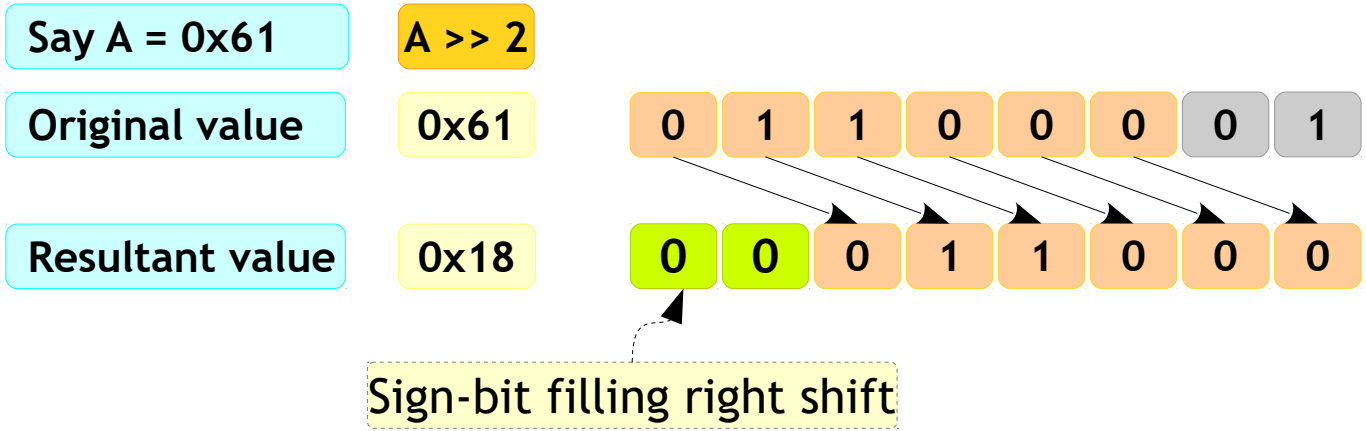
- Value : Is shift operand on which bit shifting effect to be applied
- Bits count : By how many bit(s) the given “Value” to be shifted



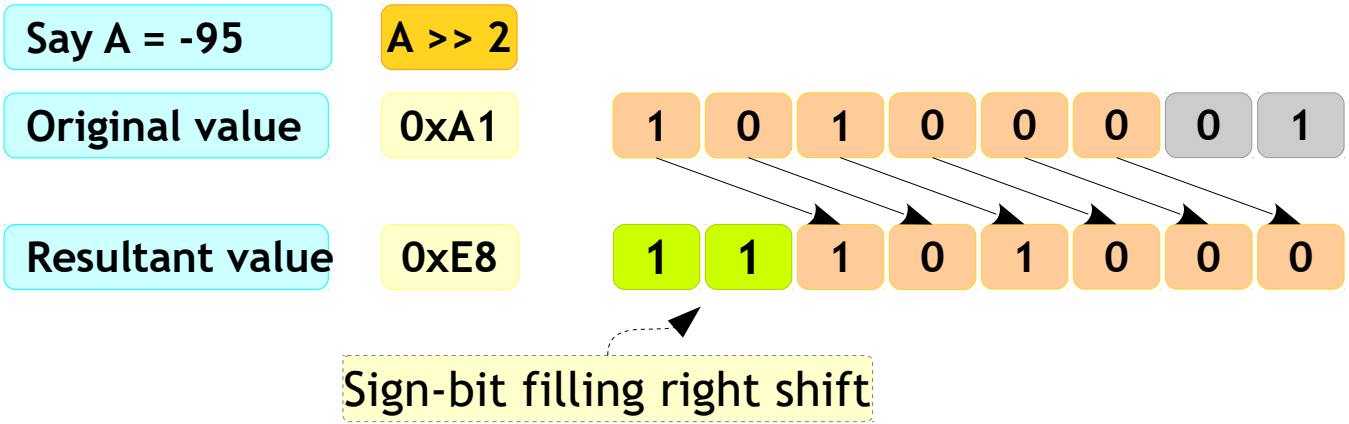
Bitwise operators

'Value' >> 'Bits Count'

- Value : Is shift operand on which bit shifting effect to be applied
- Bits count : By how many bit(s) the given “Value” to be shifted



Bitwise operators



Bitwise operator
(8-bit system examples)

Operator	Example	Same as	Result	Same as
&	5 & 1	0000 0101 & 0000 0001	1	0000 0001
|	5 | 2	0000 0101 | 0000 0010	7	0000 0111
~	~ 5	~0000 0101	-6	1111 1010
^	5 ^ 1	0000 0101 ^ 0000 0001	4	0000 0100
<<	5 << 2	0000 0101 << 2	20	0001 0100
>>	-5 >> 2	1111 1011 >> 2	-2	1111 1110
>>>	-5 >>> 2	1111 1011 >>> 2	62	0011 1110

Class Work

- WAP to print bits of a given number
- WAP to count set bits in a given number

WSA | Forward looking IT finishing school

Web Stack Academy (P) Ltd

#83, Farah Towers,

1st floor,MG Road,

Bangalore – 560001

M: +91-80-4128 9576

T: +91-98862 69112

E: info@www.webstackacademy.com