JS

Basic Fundamentals Of JavaScript

ts thruskills

https://www.thruskills.com

+91 831 737 5392

What you will learn today

- ✓ History Of JavaScript
- ✓ Writing JavaScript Code
- ✓ Language Fundamentals
- ✓ How To Think - JavaScript

History Of JavaScript

1995

JavaScript was invented by **Brendan Eich**

1997

ECMAScript 1: JavaScript became an ECMA standard

1999

ECMAScript 3: Added Regular Expressions & try/catch

2009

ECMAScript 5: Added "strict mode" & JSON support

2015

ECMAScript 6: Added classes and modules

2016

ECMAScript 7: Added exponential operator & Array.prototype.includes

Writing JavaScript Code

- ✓ Statements
- ✓ Comments
- ✓ Assignments & Equality
- ✓ Expressions

Writing JavaScript Code

Statements

```javascript
var car = "Jaguar"; // Assign the text "Jaguar" to the variable car.
var today = new Date(); // Assign today's date to the variable today.
```

Comments

```javascript
// This is a single-line comment.

/*
This is a multiline comment that explains the preceding code statement.
To comment multiple line at the same time
*/
```

Assignments and Equality

```javascript
var name = "manohar"; //assigning the value "manohar" to name

if(name === "manohar"){....
```

Writing JavaScript Code...

Expressions

JavaScript Literal Expression

```
3.9                          // numeric literal
"Hello!"                     // string literal
false                        // boolean literal
null                         // literal null value
{x:1, y:2}                   // Object literal
[1,2,3]                      // Array literal
function(x){return x*x;}     // function literal
```

JavaScript Complex Expression

```
var anExpression = 3 * (4 / 5) + 6;
var aSecondExpression = Math.PI * radius * radius;
var aThirdExpression = aSecondExpression + "%" + anExpression;
var aFourthExpression = "(" + aSecondExpression + ") % (" + anExpression + ")";
```

Language Fundamentals

- ✓ Statements
- ✓ Comments
- ✓ Assignments
- ✓ Variables
- ✓ Literals
- ✓ Keywords
- ✓ Data Types
- ✓ Operators
- ✓ Events
- ✓ Functions
- ✓ Arrays
- ✓ Objects

Language Fundamentals

{}

Statements

statements combine expressions in such a way that they carry out one complete task

var

Variables

a variable contains a value, such as "hello" or 5

if

Control Flow

transfer of program control is based upon a decision

//

Comments

*a **comment** is a **programmer**-readable explanation*

“”

Data Types

classification of data which tells the compiler or interpreter the intended to use the data.

()

Functions

functions combine several operations under one name.

=

Assignments

equal sign (=) is used in JavaScript statements to assign values

+ -

Operators

a symbol that tells the compiler to perform specific mathematical or logical manipulations

[]

Arrays

an array is a collection of elements of similar data types

new

Keywords

keywords** are tokens that have special meaning in **JavaScript

click

Events

Events are actions or occurrences that happen in the system you are programming

{}

Objects

JavaScript objects are collections of properties and methods.

Variables

Declaration

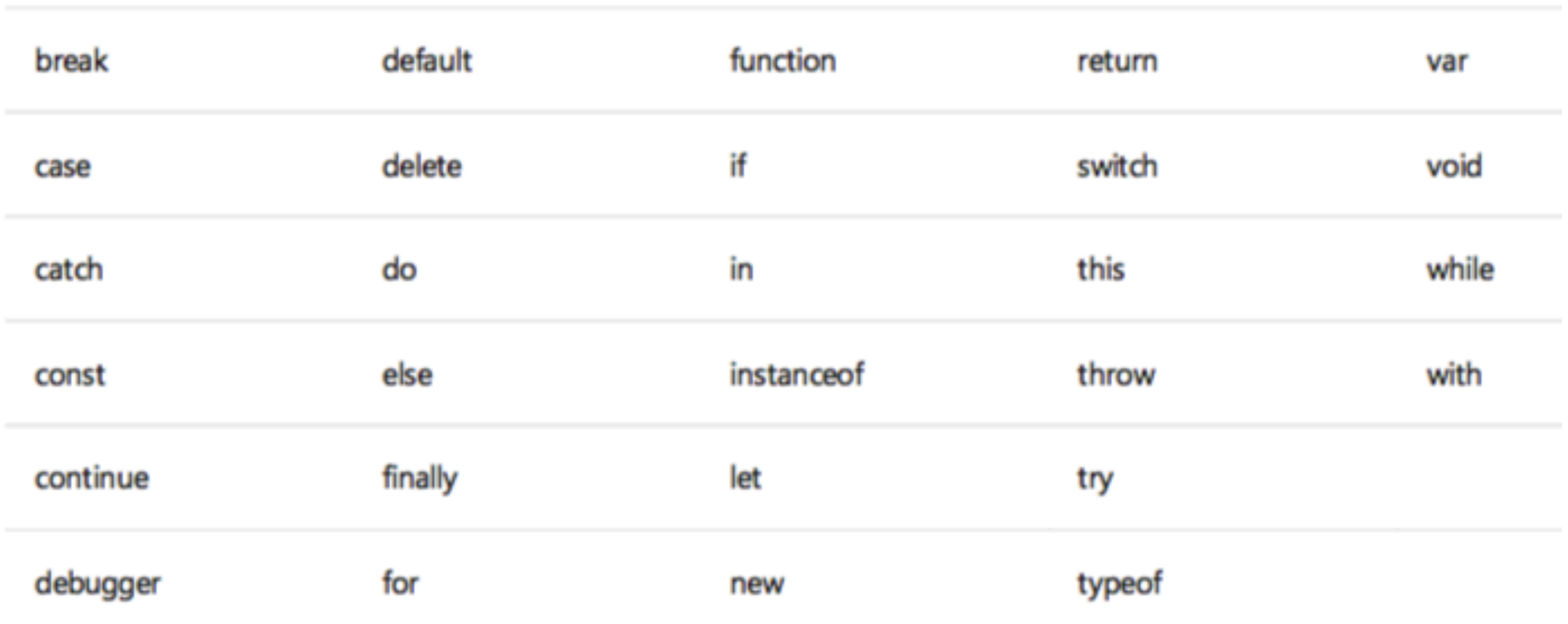
```
var car = "Jaguar"; // Assign the text "Jaguar" to the variable car.
var today = new Date(); // Assign today's date to the variable today.
```

Naming

1. JavaScript is a case-sensitive language
2. The first character must be an ASCII letter (either uppercase or lowercase), or an underscore (_) character
3. Number cannot be used as the first character
4. Subsequent characters must be letters, numbers, or underscores (_)
5. The variable name must not be a reserved word

Keywords

Reserved Words



break	default	function	return	var
case	delete	if	switch	void
catch	do	in	this	while
const	else	instanceof	throw	with
continue	finally	let	try	
debugger	for	new	typeof	

Future Reserved Words

class enum export extends import super

Variables...

Coercion

1. JavaScript is a loosely typed language, as opposed to strongly typed languages like C++
2. In JavaScript, you can perform operations on values of different types without causing an exception
3. The JavaScript interpreter implicitly converts, or coerces, one of the data types to that of the other, then performs the operation

Rules Of Coercion

1. If you add a number and a string, the number is coerced to a string
2. If you add a Boolean and a string, the Boolean is coerced to a string
3. If you add a number and a Boolean, the Boolean is coerced to a number
4. In the following example, a number added to a string results in a string

```
var x = 2000;
var y = "Hello";
// The number is coerced to a string.
x = x + y;
document.write(x);

// Output:
// 2000Hello
```

Data Types

Primary Data Types

String Number Boolean

Composite Data Types

Object Array

Special Data Types

undefined null

Data Types…

String Data Type

```
"Happy am I; from care I'm free!"
'"Avast, ye lubbers!" roared the technician.'
"45"
'c'
```

Special Characters

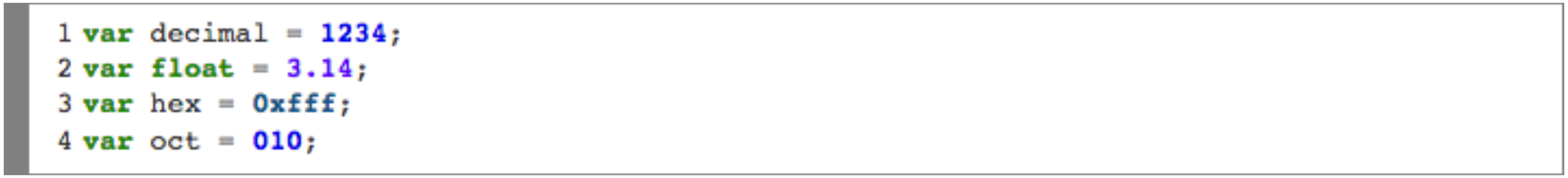
- Escape Sequences
- Unicode Code Point Escape Sequences

```
document.write('The image path is C:\\webstuff\\mypage\\gifs\\garden.gif.');
document.write ("<br />");
document.write('The caption reads, "After the snow of \'97. Grandma\'s house is covered
```

```
"\u{20BB7}"=="𠮷"=="\uD842\uDFB7"
```

Data Types...

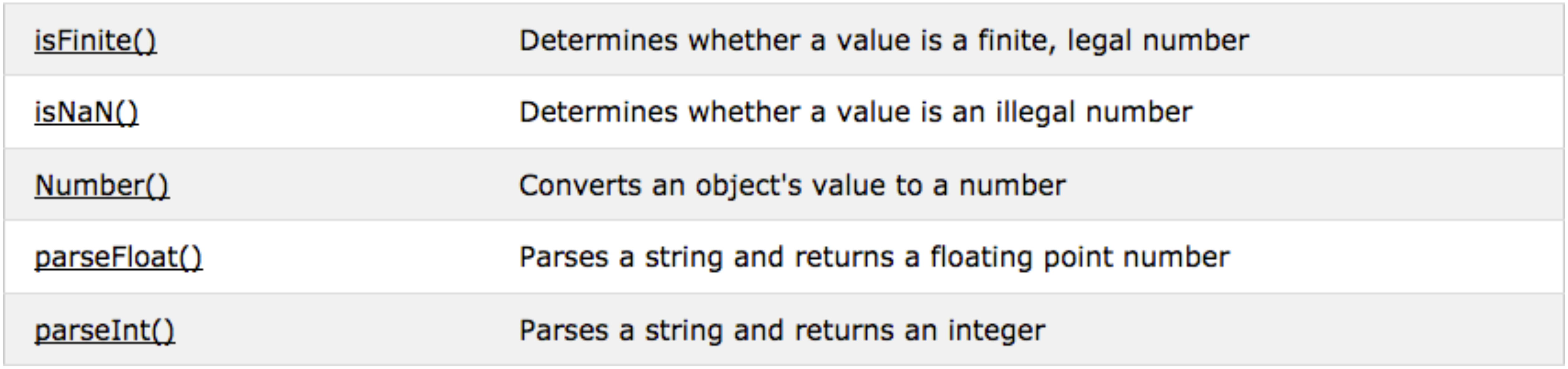
Number Data Type



```
var decimal = 1234;
var float = 3.14;
var hex = 0xfff;
var oct = 010;
```

Additionally, JavaScript contains numbers with special values. These are:

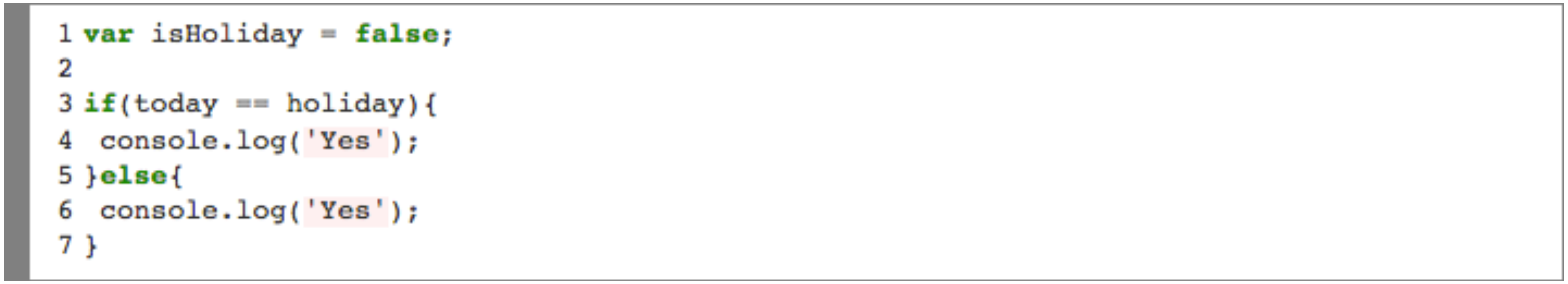
1. NaN (not a number): This is used when a mathematical operation is performed on inappropriate data, such as strings or the undefined value
2. Positive Infinity: This is used when a positive number is too large to represent in JavaScript
3. Negative Infinity: This is used when a negative number is too large to represent in JavaScript
4. Positive and Negative 0: JavaScript differentiates between positive and negative zero.



isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer

Data Types...

Boolean Data Type



```
var isHoliday = false;

if(today == holiday){
 console.log('Yes');
}else{
 console.log('Yes');
}
```

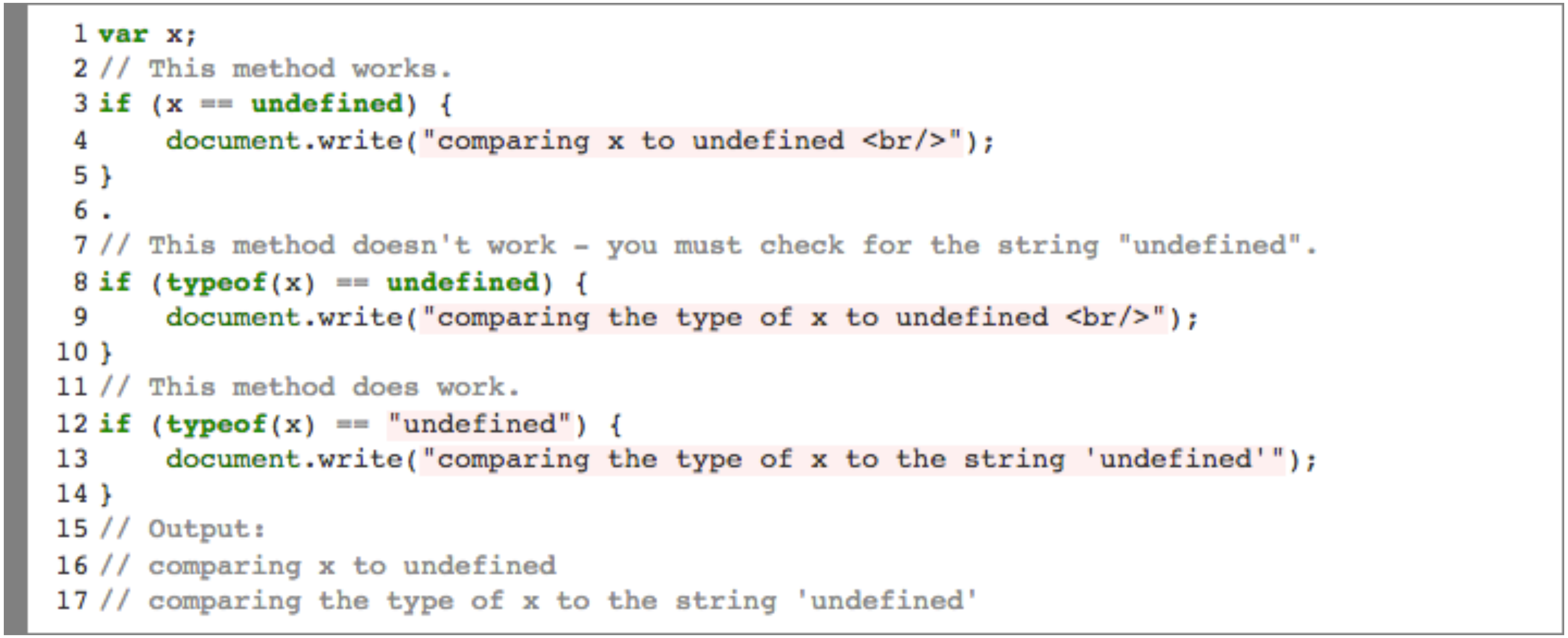
null Data Type

The null data type has only one value in JavaScript: **null**. The null keyword can not be used as the name of a function or variable.

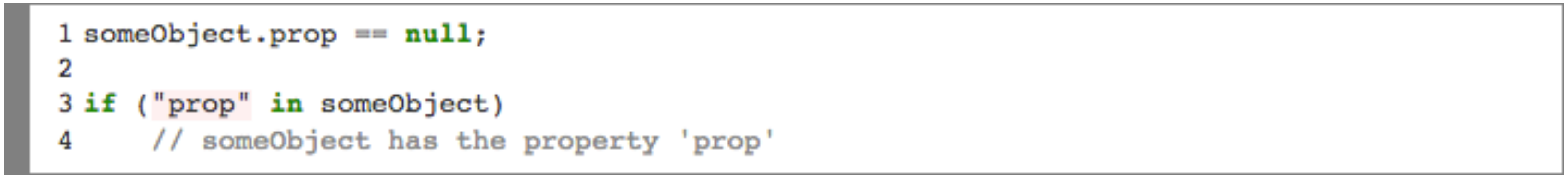
undefined Data Type

The undefined value is returned when you use an object property that does not exist, or a variable that has been declared, but has never had a value assigned to it.

Data Types...



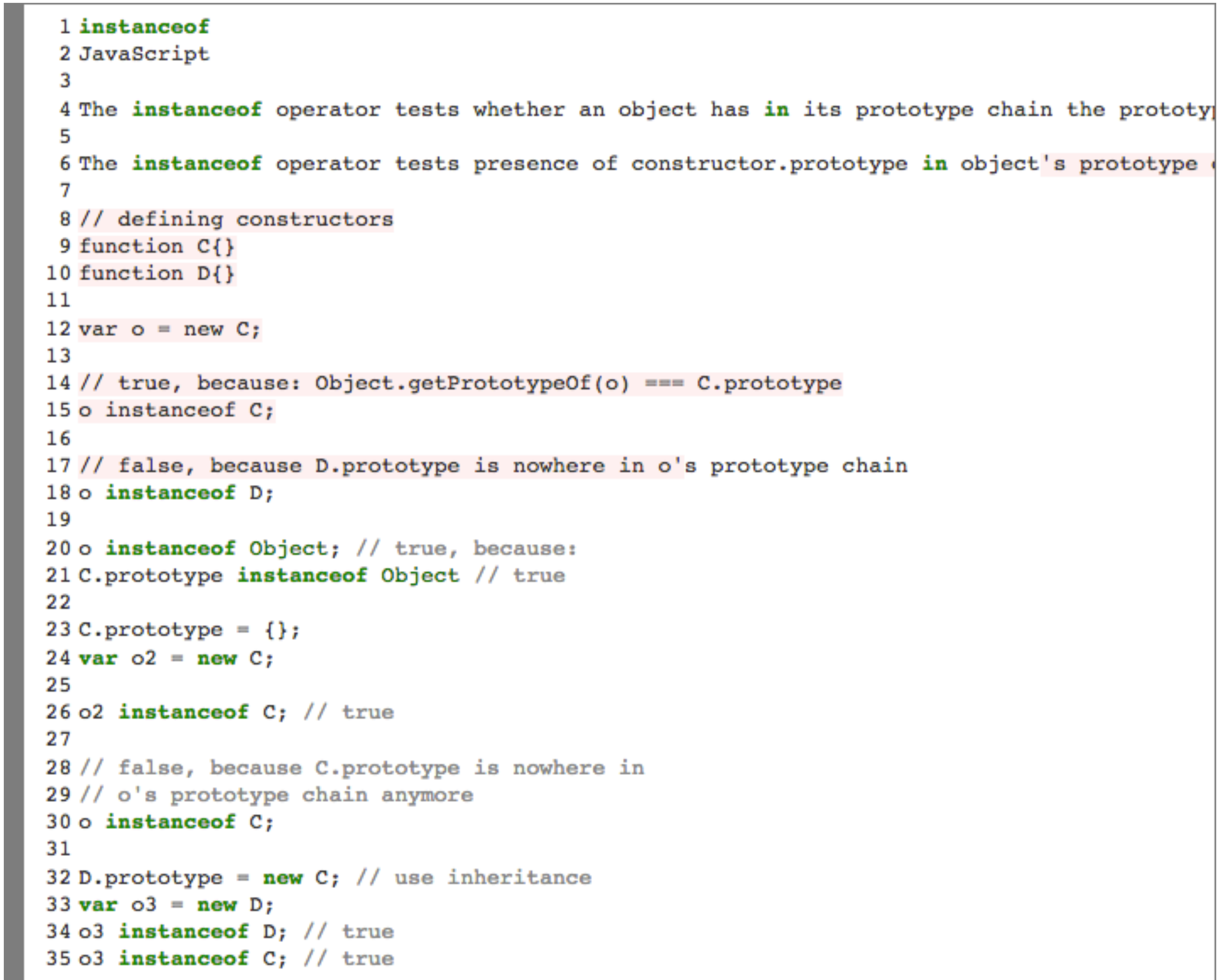
```
var x;
// This method works.
if (x == undefined) {
    document.write("comparing x to undefined <br/>");
}
.
// This method doesn't work - you must check for the string "undefined".
if (typeof(x) == undefined) {
    document.write("comparing the type of x to undefined <br/>");
}
// This method does work.
if (typeof(x) == "undefined") {
    document.write("comparing the type of x to the string 'undefined'");
}
// Output:
// comparing x to undefined
// comparing the type of x to the string 'undefined'
```



```
someObject.prop == null;

if ("prop" in someObject)
    // someObject has the property 'prop'
```

Data Types...

instanceof



```
instanceof
JavaScript

The instanceof operator tests whether an object has in its prototype chain the prototy

The instanceof operator tests presence of constructor.prototype in object's prototype 

// defining constructors
function C{}
function D{}

var o = new C;

// true, because: Object.getPrototypeOf(o) === C.prototype
o instanceof C;

// false, because D.prototype is nowhere in o's prototype chain
o instanceof D;

o instanceof Object; // true, because:
C.prototype instanceof Object // true

C.prototype = {};
var o2 = new C;

o2 instanceof C; // true

// false, because C.prototype is nowhere in
// o's prototype chain anymore
o instanceof C;

D.prototype = new C; // use inheritance
var o3 = new D;
o3 instanceof D; // true
o3 instanceof C; // true
```

Array

- ✓ Create Array
- ✓ Accessing & Modifying Array Items
- ✓ Array Length
- ✓ Array Methods

Data Types...

Array

Creating an Array

```
var shopping = ['bread', 'milk', 'cheese', 'hummus', 'noodles'];

shopping;

var sequence = [1, 1, 2, 3, 5, 8, 13];
var random = ['tree', 795, [0, 1, 2]];
```

Accessing and modifying array items

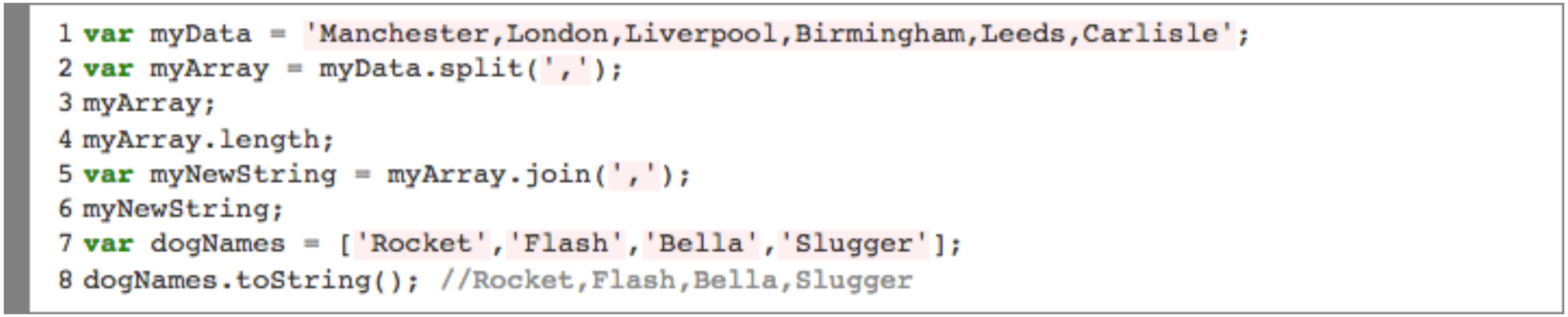
```
shopping[0]; // returns "bread"

shopping[0] = 'tahini';
shopping;
```

Finding the length of an array

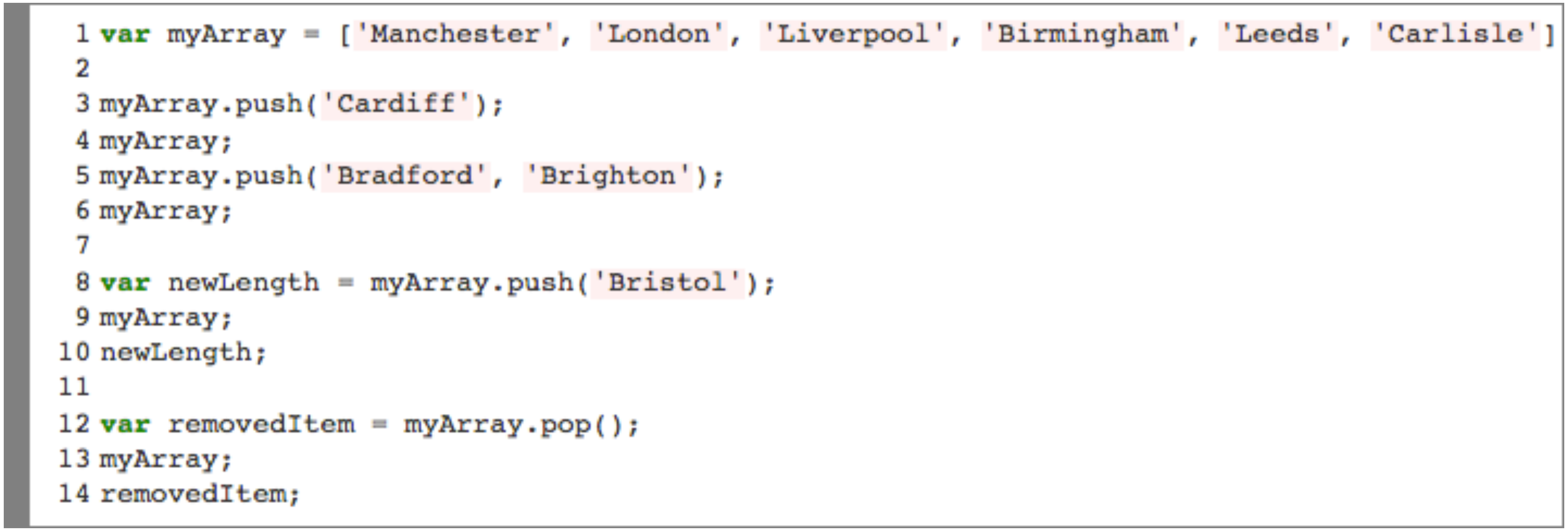
```
sequence.length; // should return 7

var sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++) {
  console.log(sequence[i]);
}
```

Data Types...

Some useful array methods



```
var myData = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';
var myArray = myData.split(',');
myArray;
myArray.length;
var myNewString = myArray.join(',');
myNewString;
var dogNames = ['Rocket','Flash','Bella','Slugger'];
dogNames.toString(); //Rocket,Flash,Bella,Slugger
```



```
var myArray = ['Manchester', 'London', 'Liverpool', 'Birmingham', 'Leeds', 'Carlisle']

myArray.push('Cardiff');
myArray;
myArray.push('Bradford', 'Brighton');
myArray;

var newLength = myArray.push('Bristol');
myArray;
newLength;

var removedItem = myArray.pop();
myArray;
removedItem;
```

Operators

- ✓ Assignment
- ✓ Comparison
- ✓ Arithmetic
- ✓ Bitwise
- ✓ Logical
- ✓ String operators
- ✓ Ternary
- ✓ Comma operator
- ✓ Unary operators
- ✓ Relational operators

Assignment Operators

= assignment operator

Name	Shorthand operator	Meaning
Assignment	`x = y`	`x = y`
Addition assignment	`x += y`	`x = x + y`
Subtraction assignment	`x -= y`	`x = x - y`
Multiplication assignment	`x *= y`	`x = x * y`
Division assignment	`x /= y`	`x = x / y`
Remainder assignment	`x %= y`	`x = x % y`
Left shift assignment	`x <<= y`	`x = x << y`
Right shift assignment	`x >>= y`	`x = x >> y`
Unsigned right shift assignment	`x >>>= y`	`x = x >>> y`
Bitwise AND assignment	`x &= y`	`x = x & y`
Bitwise XOR assignment	`x ^= y`	`x = x ^ y`
Bitwise OR assignment	`x |= y`	`x = x | y`

Comparison Operators

Operator	Description	Examples returning true

Equal (`==`)	Returns `true` if the operands are equal.	`3 == var1`
		`"3" == var1`
		
		`3 == '3'`

Not equal (`!=`)	Returns `true` if the operands are not equal.	`var1 != 4`
		`var2 != "3"`

Strict equal (`===`)	Returns `true` if the operands are equal and of the same type. See also `Object.is` and sameness in JS.	`3 === var1`

Strict not equal (`!==`)	Returns `true` if the operands are of the same type but not equal, or are of different type.	`var1 !== "3"`
		`3 !== '3'`

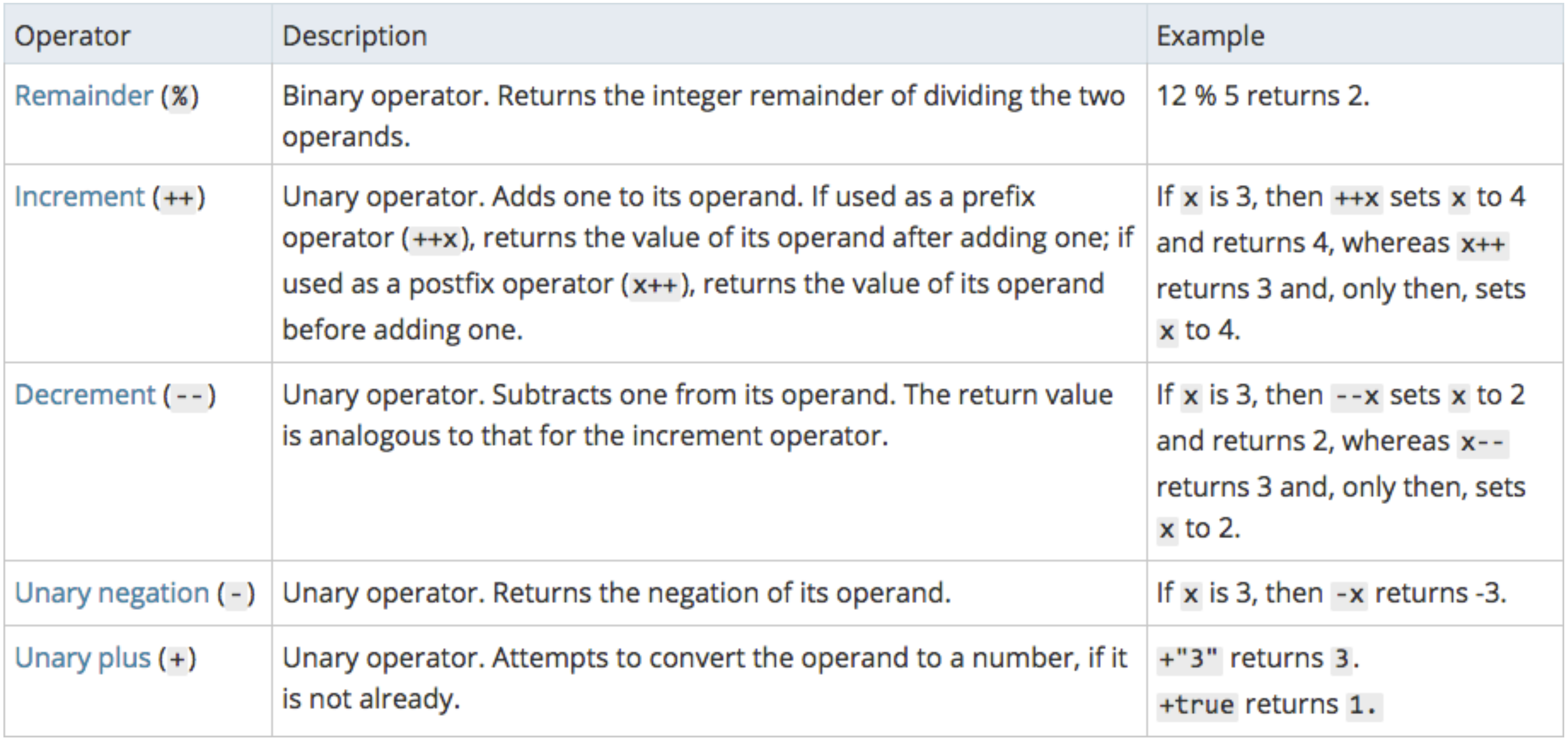
Greater than (`>`)	Returns `true` if the left operand is greater than the right operand.	`var2 > var1`
		`"12" > 2`

Greater than or equal (`>=`)	Returns `true` if the left operand is greater than or equal to the right operand.	`var2 >= var1`
		`var1 >= 3`

Less than (`<`)	Returns `true` if the left operand is less than the right operand.	`var1 < var2`
		`"2" < 12`

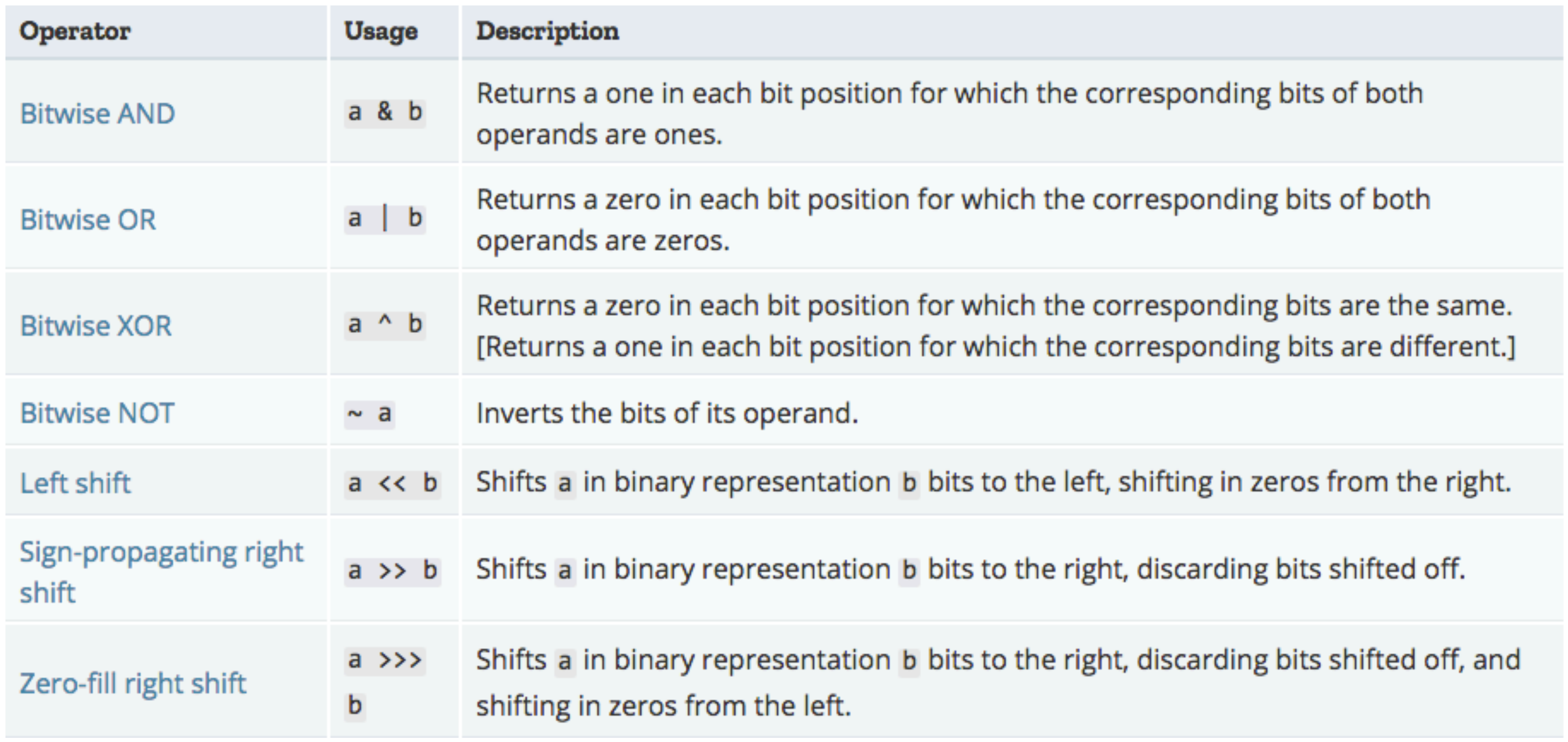
Less than or equal (`<=`)	Returns `true` if the left operand is less than or equal to the right operand.	`var1 <= var2`
		`var2 <= 5`

Arithmetic Operators



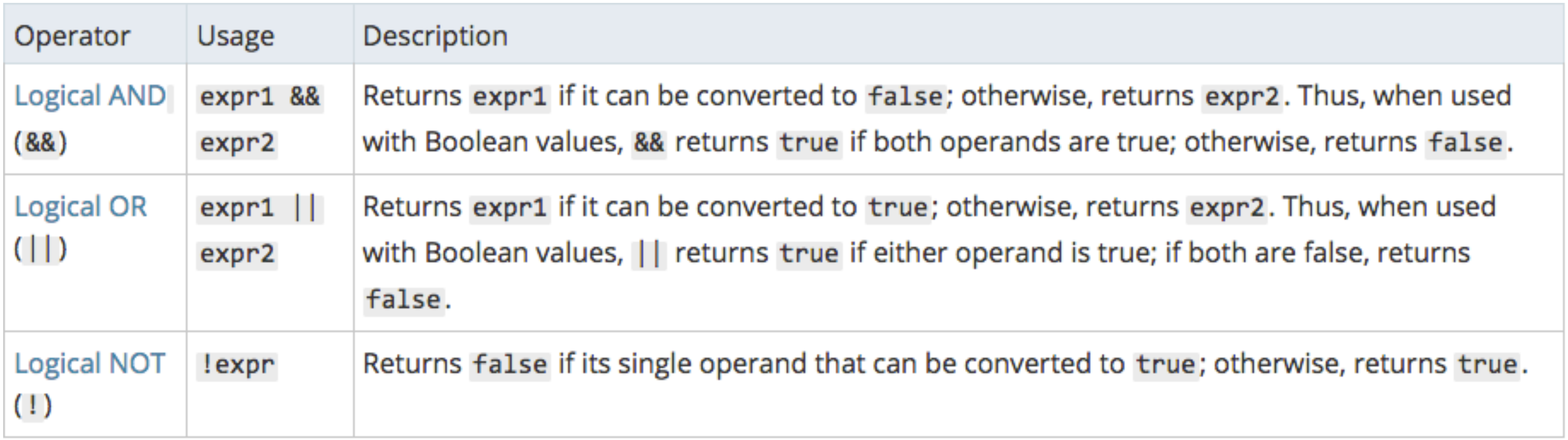
Operator	Description	Example
Remainder (`%`)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
Increment (`++`)	Unary operator. Adds one to its operand. If used as a prefix operator (`++x`), returns the value of its operand after adding one; if used as a postfix operator (`x++`), returns the value of its operand before adding one.	If `x` is 3, then `++x` sets `x` to 4 and returns 4, whereas `x++` returns 3 and, only then, sets `x` to 4.
Decrement (`--`)	Unary operator. Subtracts one from its operand. The return value is analogous to that for the increment operator.	If `x` is 3, then `--x` sets `x` to 2 and returns 2, whereas `x--` returns 3 and, only then, sets `x` to 2.
Unary negation (`-`)	Unary operator. Returns the negation of its operand.	If `x` is 3, then `-x` returns -3.
Unary plus (`+`)	Unary operator. Attempts to convert the operand to a number, if it is not already.	`+"3"` returns `3`. `+true` returns `1.`

Bitwise Operators



Operator	Usage	Description
Bitwise AND	`a & b`	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	`a | b`	Returns a zero in each bit position for which the corresponding bits of both operands are zeros.
Bitwise XOR	`a ^ b`	Returns a zero in each bit position for which the corresponding bits are the same. [Returns a one in each bit position for which the corresponding bits are different.]
Bitwise NOT	`~ a`	Inverts the bits of its operand.
Left shift	`a << b`	Shifts `a` in binary representation `b` bits to the left, shifting in zeros from the right.
Sign-propagating right shift	`a >> b`	Shifts `a` in binary representation `b` bits to the right, discarding bits shifted off.
Zero-fill right shift	`a >>> b`	Shifts `a` in binary representation `b` bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Logical Operators



Operator	Usage	Description
Logical AND (`&&`)	`expr1 && expr2`	Returns `expr1` if it can be converted to `false`; otherwise, returns `expr2`. Thus, when used with Boolean values, `&&` returns `true` if both operands are true; otherwise, returns `false`.
Logical OR (`||`)	`expr1 || expr2`	Returns `expr1` if it can be converted to `true`; otherwise, returns `expr2`. Thus, when used with Boolean values, `||` returns `true` if either operand is true; if both are false, returns `false`.
Logical NOT (`!`)	`!expr`	Returns `false` if its single operand that can be converted to `true`; otherwise, returns `true`.

String Operator

Concatenation operator (+)

```
console.log('my ' + 'string'); // console logs the string "my string".
```

```
var mystring = 'alpha';
mystring += 'bet'; // evaluates to "alphabet" and assigns this value to mystring.
```

Ternary Operator

Conditional operator (?:)

```
condition ? val1 : val2
```

```
var status = (age >= 18) ? 'adult' : 'minor';
```

Comma Operator

Comma operator (,)

The comma operator (,) simply evaluates both of its operands and returns the value of the last operand. This operator is primarily used inside a for loop, to allow multiple variables to be updated each time through the loop.

For example, if a is a 2-dimensional array with 10 elements on a side, the following code uses the comma operator to update two variables at once. The code prints the values of the diagonal elements in the array:

```
var x = [0,1,2,3,4,5,6,7,8,9]
var a = [x, x, x, x, x];

for (var i = 0, j = 9; i <= j; i++, j--)
  console.log('a[' + i + '][' + j + ']= ' + a[i][j]);
```

Unary Operators

delete operator

The delete operator deletes an object, an object's property, or an element at a specified index in an array. The syntax is:

```
delete objectName;
delete objectName.property;
delete objectName[index];
delete property; // legal only within a with statement
```

You can use the delete operator to delete variables declared implicitly but not those declared with the var statement.
If the delete operator succeeds, it sets the property or element to undefined. The delete operator returns true if the operation is possible; it returns false if the operation is not possible.

```
x = 42;
var y = 43;
myobj = new Number();
myobj.h = 4;    // create property h
delete x;       // returns true (can delete if declared implicitly)
delete y;       // returns false (cannot delete if declared with var)
delete Math.PI; // returns false (cannot delete predefined properties)
delete myobj.h; // returns true (can delete user-defined properties)
delete myobj;   // returns true (can delete if declared implicitly)
```

Unary Operators

typeof operator

The typeof operator returns a string indicating the type of the unevaluated operand. operand is the string, variable, keyword, or object for which the type is to be returned. The parentheses are optional.

```
typeof operand
typeof (operand)
```

```
var myFun = new Function('5 + 2');
var shape = 'round';
var size = 1;
var foo = ['Apple', 'Mango', 'Orange'];
var today = new Date();
```

```
typeof myFun;       // returns "function"
typeof shape;       // returns "string"
typeof size;        // returns "number"
typeof foo;         // returns "object"
typeof today;       // returns "object"
typeof doesntExist; // returns "undefined"
```

```
typeof true; // returns "boolean"
typeof null; // returns "object"
```

Unary Operators

void operator

The void operator specifies an expression to be evaluated without returning a value. expression is a JavaScript expression to evaluate. The parentheses surrounding the expression are optional, but it is good style to use them. The void operator is used in either of the following ways:

```
void (expression)
void expression
```

```
<a href="javascript:void(0)">Click here to do nothing</a>
```

```
<a href="javascript:void(document.form.submit())">
Click here to submit</a>
```

You can use the void operator to specify an expression as a hypertext link. The expression is evaluated but is not loaded in place of the current document.

Relational Operators

A relational operator compares its operands and returns a Boolean value based on whether the comparison is true.

in operator

The in operator returns true if the specified property is in the specified object. The syntax is:

```
propNameOrNumber in objectName
```

```
// Arrays
var trees = ['redwood', 'bay', 'cedar', 'oak', 'maple'];
0 in trees;        // returns true
3 in trees;        // returns true
6 in trees;        // returns false
'bay' in trees;    // returns false (you must specify the index number,
                   // not the value at that index)
'length' in trees; // returns true (length is an Array property)

// built-in objects
'PI' in Math;          // returns true
var myString = new String('coral');
'length' in myString;  // returns true

// Custom objects
var mycar = { make: 'Honda', model: 'Accord', year: 1998 };
'make' in mycar;  // returns true
'model' in mycar; // returns true
```

Relational Operators

instance operator

The instanceof operator returns true if the specified object is of the specified object type. The syntax is:

```
objectName instanceof objectType
```

```
var theDay = new Date(1995, 12, 17);
if (theDay instanceof Date) {
  // statements to execute
}
```

Use instanceof when you need to confirm the type of an object at runtime. For example, when catching exceptions, you can branch to different exception-handling code depending on the type of exception thrown.

Operator Precedence

Operator type	Individual operators
member	`. []`
call / create instance	`() new`
negation/increment	`! ~ - + ++ -- typeof void delete`
multiply/divide	`* / %`
addition/subtraction	`+ -`
bitwise shift	`<< >> >>>`
relational	`< <= > >= in instanceof`
equality	`== != === !==`
bitwise-and	`&`
bitwise-xor	`^`
bitwise-or	`|`
logical-and	`&&`
logical-or	`||`
conditional	`?:`
assignment	`= += -= *= /= %= <<= >>= >>>= &= ^= |=`
comma	`,`

Any Questions

- ✓ JavaScript Basic
- ✓ Building Blocks

Thank You

https://www.thruskills.com

+91 831 737 5392

ts thruskills