Conditional, Comma, typeof, and instanceof Operators

In this lesson, we will discover a multitude of concepts: conditional, comma, typeof, and instanceof operators. Let's begin!

Comma operator

JavaScript uses the comma operator to allow the execution of **more than one operation** in a single statement, as in this example:

index.js

```
var val1 = 1, val2 = 2, val23 = 23;
```

You can use this operator in expressions, too. In this way, the comma operator returns the value of the rightmost (last) expression, such as in this short example:

JS index.js

```
var myValue = (2, 3, 4, 5);
console.log(myValue);              // 5
console.log( 1 + ("a", "b", "c")); // 1c
```

The `typeof` and `instanceof` operators

You already learned that the `typeof` operator can be used to determine the type of a value:

JS index.js

```
console.log(typeof "hey!"); // string
```

Although `typeof` works well for primitive values, it's of little use for reference values because it retrieves `“object”` or `“function”` for all reference types.

Typically, you want to know more than the value of an object, you may also want to know its type.

For example, if you created an instance with the `Car` constructor function, you want to know that the object is an instance of `Car`.

To benefit in this identification, **ECMAScript** provides the `instanceof` operator, which is used with this syntax:

JS index.js

```
result = variable instanceof constructor
```

The `instanceof` operator returns true if the variable is an instance of the given reference type.

Take a look at this example:

index.js

```
// Is the variable car a Car?
console.log(car instanceof Car);
// Is the variable peter a Person?
console.log(peter instanceof Person);
// Is the variable list an Array?
conselo.log(list instanceof Array);
```

> 📃**NOTE:** All reference values, by definition, are instances of `Object`, so the `instanceof` operator always returns true when used with a reference value and the `Object` constructor. Similarly, if `instanceof` is used with a primitive value, it will always return false.

In the *next lesson*, we'll check out operator precedence.