Coercion and Truthiness

Overview

```
/*
   - Type
   - Explicit coercion
   - Implicit coercion
     - Concatenation
     - Loose-equality operator (==)
   - Coercion to boolean (truthiness)
     - Which values are truthey/falsey
     - conditional expressions
  - ! operator
     - logical operators */
```

Types

```
/*
Types you've already seen:
  - number
  - string
  - boolean

These are all primitive types!

Other primitive types:
  - null
  - undefined
  - Symbol (not covered in this course)
*/

```

Types

```
/*
"Complex" or "object" types:

- functions
- arrays (will cover later)
- objects (will cover later)

Major differences between primitive and complex types:

  - only complex types can be mutated (will cover later)
  - primitive types are passed by value, complex types are passed by reference (will cover later)



*/
```

Explicit coercion

```
number
10
string
```

```
1   /* Coercion occurs when the type of a value is change to a new type */
2
3   /* Explicit coercion happens when we use one of built-in global objects
4      to create a value of a new type */
5
6   let num = 10;
7   console.log(typeof num);
8
9   let string = String(num); // String global object
10  console.log(string);
11  console.log(typeof string);
12
13
14
```

Explicit coercion

```
string
1000
number
```

```
1   /* Coercion occurs when the type of a value is change to a new type */
2
3   /* Explicit coercion happens when we use one of built-in global objects
4      to create a value of a new type */
5
6   let string = '1000';
7   console.log(typeof string);
8
9   let num = Number(string); // Number global object
10  console.log(num);
11  console.log(typeof num);
12
13
14
```

Implicit coercion

```
/* implicit coercion also changes the type of a value */



/* unlike explicit coercion, implicit coercion is something that
   JavaScript does for us, behind the scenes */



/* this behavior can be very helpful, but it's important to understand how
   it works so we can anticipate what our code will do */
```

Implicit coercion: +

```
sum: 30
concatenatedString: 1020
notSure: 1020
```

```
1   let sum = 10 + 20;
2
3   let concatenatedString = '10' + '20';
4
5   let notSure = 10 + '20'; // will this throw an error? return a value?
6
7   console.log('sum', sum);
8   console.log('concatenatedString:', concatenatedString);
9   console.log('notSure:', notSure);
10
11
12
13
14
```

Implicit coercion: +

```
/* where does 1020 come from? note it's the same value as concatenating
   '10' and '20' */

/* the + operator will implicitly coerce a number to a string if you try
   to 'add' it to a string */

let willBeAString = '10' + 20 + 30 + 40 + 50;
console.log(willBeAString);
console.log(typeof willBeAString);
```

```
1020304050
string
```

10050
string

Implicit coercion: +

```
/* where does 1020 come from? note it's the same value as concatenating
   '10' and '20' */

/* the + operator will implicitly coerce a number to a string if you try
   to 'add' it to a string */

let alsoAString = 10 + 20 + 30 + 40 + '50';
console.log(alsoAString);
console.log(typeof alsoBeAString);
```

Implicit coercion: ==

```
/* avoid using the == operator, because it uses a large set of rules to
   implicitly coerce values to the same type before comparing them. */

10 == 10; // => true, makes sense
10 == '10'; // => true, also makes sense
'true' == true; // => false, kinda weird
'' == false; // => true, kinda weird
true == '1'; // => true, kinda weird*

/* *behind the scenes, JS coerced both of these values to numbers:
   true coerced to 1
   '1' coerced to  1
   1 == 1 => true
*/
```

true

Boolean coercion: truthiness

```
1   // values can be coerced to boolean values, too
2
3   let newBool = Boolean('i am a string');
4
5   console.log(newBool); // will this be true or false?
6
7
8
9
10
11
12
13
14
```

Boolean coercion: truthiness

```
/* when coercing a value to boolean, JS uses rules to decide if a value
   should be coerced to true or false */



/* values coerced to true are called "truthey" */



/* values coerced to false are called "falsey" */
```

Boolean coercion: truthiness

```
true
true
true
true
```

```
/* Most values are truthy */

console.log(Boolean('i am a string')) // strings with length are truthy

console.log(Boolean(10)); // any non-zero number is truthy

console.log(Boolean(['i', 'am', 'an', 'array'])); // all arrays are truthy

console.log(Boolean({
    i: 'am',
    an: 'object'
})); // all objects are truthy


```

Boolean coercion: truthiness

```
1   /* These are the only falsey values */
2
3   console.log(Boolean('')) // empty string
4
5   console.log(Boolean(0));
6
7   console.log(Boolean(null));
8
9   console.log(Boolean(undefined));
10
11  console.log(Boolean(NaN));
12
13
14
```

```
false
false
false
false
false
```

Boolean coercion: conditionals

```
1   /* Recall how a conditional expression works in an if statement */
2
3   // if the expression below evaluates to true, the if block will run
4   if (5) {
5     console.log('in the if');
6   }
7   else {
8     console.log('in the else');
9   }
10
11
12
13
14
```

Boolean coercion: conditionals

```
1   /* What if the conditional expression evaluates to a non-boolean value? */
2
3   if ('apples') {
4     console.log('in the if');
5   }
6   else {
7     console.log('in the else');
8   }
9
10
11
12
13
14
```

Boolean coercion: conditionals

```
1   /* JS will implicitly coerce the result of an expression in a conditional
2      to a boolean value */
3   if (10) {
4     console.log('yes');
5   }
6   else {
7     console.log('no');
8   }
9
10  if (0) {
11    console.log('yes');
12  else {
13    console.log('no');
14  }
```

```
yes
no
```

Boolean coercion: ! operator

```
/* The ! operator coerces a value to a boolean value that's opposite of
   its truthiness (that's why ! is also called the not operator) */

console.log(!true); = Opposite of what Boolean(true) evaluates to

console.log(!'abc'); = opposite of what Boolean(‘abc’) evaluates to

console.log(!100); = opposite of what Boolean(100) evaluates to

console.log(!['an', 'array']); = opposite of what Boolean([“an”, “array”]) evaluates to

console.log(!{an: 'object'}); = opposite of what Boolean({“an”: “object”}) evaluates to


```

```
false
false
false
false
false
```

Boolean coercion: !! operator

```
true
true
true
true
true
```

```
1   /* You can use !! to explicitly coerce a value to a boolean value that
2      reflects its truthiness (not not) */
3
4   console.log(!!true);
5
6   console.log(!!'abc');
7
8   console.log(!!100);
9
10  console.log(!!['an', 'array']);
11
12  console.log(!!{an: 'object'});
13
14
```

both 10 and 20 are truthy values

Logical operators

```
1   /* Logical operators also coerce values to boolean values */
2
3   if (10 && 20) {
4     console.log('both 10 and 20 are truthy values');
5   }
6
7
8
9
10
11
12
13
14
```

zero is falsey

Logical operators

```
1   /* Logical operators also coerce values to boolean values */
2
3   if (10 && 0) {
4     console.log('this will not be logged');
5   }
6   else {
7     console.log('zero is falsey');
8   }
9
10
11
12
13
14
```

Logical operators

```
returnedValue1: 0
returnedValue2: 7
```

```
/* && returns the first falsey value, or the last value if all are
   truthy */

let returnedValue1 = 10 && 'apples' && 0 && null;
console.log('returnedValue1:', returnedValue1);

let returnedValue2 = 'lucky' && 'number' && 7;
console.log('returnedValue2:', returnedValue2);
```

Logical operators

```
returnedValue1: happy
returnedValue2: NaN
```

```
/* || returns the first truthy value, or the last value if all are
   falsey */

let returnedValue1 = null || undefined || 'happy' || 'pumpkin';
console.log('returnedValue1:', returnedValue1);

let returnedValue2 = false || null || 10 < 0 || NaN;
console.log('returnedValue2:', returnedValue2);
```

Recap

```
/*
  - Types

  - Explicit coercion

  - Implicit coercion
    - Concatenation
    - Loose-equality operator (==)

  - Coercion to boolean (truthiness)
    - Which values are truthy/falsey
    - conditional expressions
- ! operator
    - logical operators */
```