

Coercion and Truthiness



Overview

- 1 /*
- 2 - Type
- 3 - Explicit coercion
- 4 - Implicit coercion
- 5 - Concatenation
- 6 - Loose-equality operator (==)
- 7 - Coercion to boolean (truthiness)
- 8 - Which values are truthy/falsey
- 9 - conditional expressions
- 10 - ! operator
- 11 - logical operators */
- 12
- 13
- 14



Types

```
1  /*
2  Types you've already seen:
3    - number
4    - string
5    - boolean
6
7  These are all primitive types!
8
9  Other primitive types:
10   - null
11   - undefined
12   - Symbol (not covered in this course)
13  */
14
```



Types

```
1  /*
2  "Complex" or "object" types:
3
4  - functions
5  - arrays (will cover later)
6  - objects (will cover later)
7
8  Major differences between primitive and complex types:
9
10 - only complex types can be mutated (will cover later)
11 - primitive types are passed by value, complex types are passed by reference (will cover later)
12
13
14  */
```



number
10
string

Explicit coercion

```
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
```



string
1000
number

Explicit coercion

```
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

```
1  /* implicit coercion also changes the type of a value */
2
3
4
5  /* unlike explicit coercion, implicit coercion is something that
6     JavaScript does for us, behind the scenes */
7
8
9
10 /* this behavior can be very helpful, but it's important to understand how
11    it works so we can anticipate what our code will do */
12
13
14
```



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
```




1020304050
string

Implicit coercion: +

```
1  /* where does 1020 come from? note it's the same value as concatenating
2   * '10' and '20' */
3
4  /* the + operator will implicitly coerce a number to a string if you try
5   * to 'add' it to a string */
6
7  let willBeAString = '10' + 20 + 30 + 40 + 50;
8  console.log(willBeAString);
9  console.log(typeof willBeAString);
10
11
12
13
14
```



10050
string

Implicit coercion: +

```
1 /* where does 1020 come from? note it's the same value as concatenating
2    '10' and '20' */
3
4 /* the + operator will implicitly coerce a number to a string if you try
5    to 'add' it to a string */
6
7 let alsoAString = 10 + 20 + 30 + 40 + '50';
8 console.log(alsoAString);
9 console.log(typeof alsoBeAString);
10
11
12
13
14
```



Implicit coercion: ==

```
1  /* avoid using the == operator, because it uses a large set of rules to
2     implicitly coerce values to the same type before comparing them. */
3
4  10 == 10; // => true, makes sense
5  10 == '10'; // => true, also makes sense
6  'true' == true; // => false, kinda weird
7  '' == false; // => true, kinda weird
8  true == '1'; // => true, kinda weird*
9
10 /* *behind the scenes, JS coerced both of these values to numbers:
11     true coerced to 1
12     '1' coerced to 1
13     1 == 1 => true
14 */
```



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

```
1  /* when coercing a value to boolean, JS uses rules to decide if a value
2     should be coerced to true or false */
3
4
5
6  /* values coerced to true are called "truthy" */
7
8
9
10 /* values coerced to false are called "falsey" */
11
12
13
14
```



true
true
true
true

Boolean coercion: truthiness

```
1  /* Most values are truthy */
2
3  console.log(Boolean('i am a string')) // strings with length are truthy
4
5  console.log(Boolean(10)); // any non-zero number is truthy
6
7  console.log(Boolean(['i', 'am', 'an', 'array'])); // all arrays are truthy
8
9  console.log(Boolean({
10    i: 'am',
11    an: 'object'
12  })); // all objects are truthy
13
14
```



Boolean coercion: truthiness

false
false
false
false
false

```
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
```



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
```



yes
no

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 }
13 else {
14     console.log('no');
15 }
```



false
false
false
false
false

Boolean coercion: ! operator

```
1  /* The ! operator coerces a value to a boolean value that's opposite of
2     its truthiness (that's why ! is also called the not operator) */
3
4  console.log(!true); = Opposite of what Boolean(true) evaluates to
5
6  console.log(!'abc'); = opposite of what Boolean('abc') evaluates to
7
8  console.log(!100); = opposite of what Boolean(100) evaluates to
9
10 console.log(!['an', 'array']); = opposite of what Boolean(["an", "array"]) evaluates to
11
12 console.log(!{an: 'object'}); = opposite of what Boolean({"an": "object"}) evaluates to
13
14
```



true
true
true
true
true

Boolean coercion: !! operator

```
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
```



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
```



returnedValue1: 0
returnedValue2: 7

Logical operators

```
1  /* && returns the first falsey value, or the last value if all are
2     truthy */
3
4  let returnedValue1 = 10 && 'apples' && 0 && null;
5  console.log('returnedValue1:', returnedValue1);
6
7  let returnedValue2 = 'lucky' && 'number' && 7;
8  console.log('returnedValue2:', returnedValue2);
9
10
11
12
13
14
```



returnedValue1: happy
returnedValue2: NaN

Logical operators

```
1  /* || returns the first truthy value, or the last value if all are
2     falsey */
3
4  let returnedValue1 = null || undefined || 'happy' || 'pumpkin';
5  console.log('returnedValue1:', returnedValue1);
6
7  let returnedValue2 = false || null || 10 < 0 || NaN;
8  console.log('returnedValue2:', returnedValue2);
9
10
11
12
13
14
```




Recap

- 1 /*
- 2 - Types
- 3 - Explicit coercion
- 4
- 5 - Implicit coercion
- 6 - Concatenation
- 7 - Loose-equality operator (==)
- 8
- 9 - Coercion to boolean (truthiness)
- 10 - Which values are truthy/falsey
- 11 - conditional expressions
- 12 - ! operator
- 13 - logical operators */
- 14