Start Recording for: *Coercion*

• REC

Changes in Type

- Coercion occurs when a value changes (is "coerced") from one type to another.

- Recall the built in types in JavaScript (ES5):
  1. null
  2. undefined
  3. boolean
  4. number
  5. string
  6. object

```
var myNum = 4;
var myString = "8";

var combined = myNum + myString;
console.log(combined)              // 48
console.log(typeof combined)       // string
console.log(typeof myNum)          // number


/*

How did adding myNumber produce a string?

When "added", the underlying value was
first converted to a string and then
concatenated to myString.

This did NOT change the type of myNum

*/
```

When does Coercion happen?

- Coercion occurs in two primary places:
  - 1. Operations
    - `myNum + myStr`
  - 2. Test Expressions
    - `if (bool) { … }`
- Coercion *always* results in a primitive value.
  - *Operations* can result in any primitive.
  - *Test expressions* will coerce to a boolean

```
// Coercion with operations
var strOne = "1"
var strTwo = "2"
var sum = strOne*2 + strTwo;

// Multiplication operation coerces to num
// "+" operation coerces back to string
console.log(sum)   // 22
```

```
var myStr = "hello world";

// Expression in 'if' statement is
// a 'test expression', coerced to boolean
if (myStr) {
    console.log("coerced to true");
} else {
    console.log("coerced to false");
}  // logs: coerced to true
```

Implicit vs. Explicit

- “explicit coercion” is when it is obvious from looking at the code that a type conversion is intentionally occurring
- “implicit coercion” is when the type conversion will occur as a less obvious side effect of some other intentional operation.

```
var a = 42;

var b = a + "";           // implicit coercion

var c = String( a );      // explicit coercion
```

Knowing the Result of Coercion

- Coercion rules are set by the ECMA Script Specification.
- Don't focus on trying to memorize every possible permutation of coercion. Instead, understand the process exists, and use the `typeof` operator to check a value if you're unsure.
- This lecture will focus on coercion that results in a boolean value. This is the kind of coercion that occurs in test expressions:
  - If blocks, while blocks, for blocks, ternary expressions

Truthy/Falsey

underlying boolean value

Coerced to Boolean

- Every JavaScript value and expression can be coerced to a boolean.
- Values that coerce to `true` are referred to as "truthy". Those that coerce to `false` are "falsey".
- If the interpreter expects a boolean it will coerce your value to one.

```
var myStr = "false";
var myNum = 12;
var myArr = [1, 2, 3];
var myNull = null;
var myUndefined = undefined;

// In each of the following instances, the
// interpreter expects a boolean

if (myArr) { … }

while (myStr) { … }

for (var i=0; myNum; myNum--) { … }

myNull || myUndefined && myStr

// if the value is not already a boolean
// it will be coerced to one
```

! (logical NOT)

- `!` is the 'logical NOT' operator (also called the 'bang' operator).
- It converts whatever value follows to boolean, and then swaps `true` to `false` and vice versa.
- Accordingly, using `!!` before a value will coerce the value to it's boolean.

```
// The 'bang' operator toggles the boolean following
var trueBool = true;
var falseBool = false;

console.log(!trueBool)    // false
console.log(!falseBool)    // true

// If the value that follows is NOT a boolean
// The 'bang' operator first coerces it to boolean

console.log(!0)    // true
console.log(!"hello world")    // false
```

```
// Using the ! operator "bang bang" (one right after
// the other) will reveal the underlying boolean
// value for any term
console.log(!!"hello world")   // true

// therefore we can say that the string "hello world"
// is a truth value
```

truthy or falsey?

- There is a simple way to know whether a value is truthy or falsey.

- The following values are falsey:

1. `false`
2. `0`
3. `''` and `""`
4. `null`
5. `undefined`
6. `NaN`

Everything else is truthy!

```
/*
    Falsey values in JS:
*/

console.log(!!false)    // false
console.log(!!0)    // false
console.log(!!"")    // false
console.log(!!null)    // false
console.log(!!undefined)    // false
console.log(!!NaN)    // false
```

```
/*
   All other values are truthy!!
*/

console.log(!!true)    // true
console.log(!!-1)    // true
console.log(!!"false")    // true
console.log(!![null])    // true
```

How can we use this?

- Now we can make our 'test expressions' more concise.
- For example there's no reason to test whether a value `=== 0` or whether a string is empty.

```
/*
    Old way to log even values
*/

for (var i=0; i<10; i++) {
    if (i % 2 === 0) {
         console.log("value is even!");
     }
}
```

```
/*
    Taking advantage of coercion and truthy falsey
*/

for (var i=0; i<10; i++) {
    if (!(i % 2)) {
         console.log("value is even!");
     }
}

// If even, i % 2 is 0 (falsey). So precede it
// with the bang operator to get truthy
```

Quick Practice

```
/*
What would the following expressions log out?
*/

!!5
!!(4 % 2)
!!(undefined)
!!("a".length - 1)
!!([false])
!!([])
```