JavaScript:

The fairly odd parts — explained

@bmeurer

@v8js

JavaScript:

The fairly odd parts

Benedikt Meurer
@bmeurer

"JavaScript is the new language of choice for many applications, and it's certainly a great language for many tasks. But it also has some strange quirks. This lightning talk focuses on some of JavaScript's weird parts, that you usually don't run into... unless you try to implement a JavaScript VM that correctly follows the language specification."

-- YGLF Kiyv 2018

```
16
// → 16
```

```
16
// → 16

016
// → 14
```

```
16
// → 16

016
// → 14

017
// → 15
```

```
16
// → 16

016
// → 14

017
// → 15

018
// → 18
```

```
// 16
parseInt("16", 10) → 16
```

```
// 16
parseInt("16", 10) → 16

// 016
parseInt("16", 8)
```

```
// 16
parseInt("16", 10) → 16

// 016
parseInt("16", 8) → 1 * 8**1 + 6 * 8**0
```

```
// 16
parseInt("16", 10) → 16

// 016
parseInt("16", 8) → 1 * 8**1 + 6 * 8**0 → 14
```

```
// 16
parseInt("16", 10) → 16

// 016
parseInt("16", 8) → 1 * 8**1 + 6 * 8**0 → 14

// 017
parseInt("17", 8) → 1 * 8**1 + 7 * 8**0 → 15
```

```
// 16
parseInt("16", 10) → 16

// 016
parseInt("16", 8) → 1 * 8**1 + 6 * 8**0 → 14

// 017
parseInt("17", 8) → 1 * 8**1 + 7 * 8**0 → 15

// 018
```

```
// 16
parseInt("16", 10) → 16

// 016
parseInt("16", 8) → 1 * 8**1 + 6 * 8**0 → 14

// 017
parseInt("17", 8) → 1 * 8**1 + 7 * 8**0 → 15

// 018
parseInt("18", 10) → 18
```

```
"use strict";

016
// SyntaxError: Octal literals are not allowed in strict mode.

017
// SyntaxError: Octal literals are not allowed in strict mode.

018
// SyntaxError: Decimals with leading zeros are not allowed
                in strict mode.
```

```
"use strict";

0o16
// → 14

0o17
// → 15

0o18
// → SyntaxError: Invalid or unexpected token
```

9007199254740993

```
9007199254740993
// → 9007199254740992
```

```
9007199254740993
// → 9007199254740992

9007199254740993 + 1 - 1
```

```
9007199254740993
// → 9007199254740992

9007199254740993 + 1 - 1
// → 9007199254740991
```

@bmeurer

```
9007199254740993
// → 9007199254740992

9007199254740993 + 1 - 1
// → 9007199254740991

9007199254740993 + 2 - 2
// → 9007199254740992
```

4.3.20 Number value

primitive value corresponding to a double-precision 64-bit binary format IEEE 754-2008 value

tc39.es/ecma262/#sec-terms-and-definitions-number-value

```
2**53-1 → 9007199254740991
// representable

2**53   → 9007199254740992
// representable
```

```
2**53-1 → 9007199254740991
// representable

2**53   → 9007199254740992
// representable

2**53+1 → 9007199254740992
// not representable, rounded down
```

```
2**53-1 → 9007199254740991
// representable

2**53   → 9007199254740992
// representable

2**53+1 → 9007199254740992
// not representable, rounded down

2**53+2 → 9007199254740994
// representable
```

More information

- *"Safe integers in JavaScript"* (Axel Rauschmayer)
  2ality.com/2013/10/safe-integers.html

- *"Double-precision floating-point format"*
  wikipedia.org/wiki/Float64

```
// 9007199254740993
parseInt("9007199254740993", 10)
```

```
// 9007199254740993 = 2**53+1
parseInt("9007199254740993", 10)
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
 → 9007199254740992 + 1 - 1
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
 → 9007199254740992 + 1 - 1
 → 9007199254740992 - 1
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
 → 9007199254740992 + 1 - 1
 → 9007199254740992 - 1
 → 9007199254740991
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
 → 9007199254740992 + 1 - 1
 → 9007199254740992 - 1
 → 9007199254740991

// 9007199254740993 + 2 - 2
parseInt("9007199254740993", 10) + 2 - 2
 → 9007199254740992 + 2 - 2
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
 → 9007199254740992 + 1 - 1
 → 9007199254740992 - 1
 → 9007199254740991

// 9007199254740993 + 2 - 2
parseInt("9007199254740993", 10) + 2 - 2
 → 9007199254740992 + 2 - 2
 → 9007199254740994 - 2
```

```
// 9007199254740993
parseInt("9007199254740993", 10)
→ 9007199254740992

// 9007199254740993 + 1 - 1
parseInt("9007199254740993", 10) + 1 - 1
 → 9007199254740992 + 1 - 1
 → 9007199254740992 - 1
 → 9007199254740991

// 9007199254740993 + 2 - 2
parseInt("9007199254740993", 10) + 2 - 2
 → 9007199254740992 + 2 - 2
 → 9007199254740994 - 2
 → 9007199254740992
```

```
0 <= null
```

```
0 <= null
// → true
```

```
0 <= null
// → true

0 < null
// → false
```

```
0 <= null
// → true

0 < null
// → false

0 == null
```

```
0 <= null
// → true

0 < null
// → false

0 == null
// → false
```

Two different algorithms

- *"Abstract Relational Comparison"* (<, >, <=, >=)
  tc39.es/ecma262/#sec-abstract-relational-comparison

- *"Abstract Equality Comparison"* (==, !=)
  tc39.es/ecma262/#sec-abstract-equality-comparison

```
// 0 <= null
0 <= null
```

```
// 0 <= null
0 <= null → Number(0) <= Number(null)
```

```
// 0 <= null
0 <= null → Number(0) <= Number(null)
          → 0 <= 0
```

```
// 0 <= null
0 <= null → Number(0) <= Number(null)
          → 0 <= 0
          → true
```

```
// 0 <= null
0 <= null → Number(0) <= Number(null)
          → 0 <= 0
          → true

// 0 < null
0 < null → Number(0) < Number(null)
         → 0 < 0
         → false
```

```
// 0 <= null
0 <= null → Number(0) <= Number(null)
          → 0 <= 0
          → true

// 0 < null
0 < null → Number(0) < Number(null)
         → 0 < 0
         → false

// 0 == null
0 == null
```

```
// 0 <= null
0 <= null → Number(0) <= Number(null)
          → 0 <= 0
          → true

// 0 < null
0 < null → Number(0) < Number(null)
         → 0 < 0
         → false

// 0 == null
0 == null → false
```

```
a = [0];
// → [0]
```

```
a = [0];
// → [0]

a == 0;
// → true
```

```
a = [0];
// → [0]

a == 0;
// → true

0 == [];
// → true
```

```
a = [0];
// → [0]

a == 0;
// → true

0 == [];
// → true

a == [];
```

```
a = [0];
// → [0]

a == 0;
// → true

0 == [];
// → true

a == [];
// → false
```

Hints 

- `[0].toString()` → `"0"`
- `Number("")` → `0`

```
x = 0 / 0;
// → NaN
```

```
x = 0 / 0;
// → NaN

a = [x];
// → [NaN]
```

```
x = 0 / 0;
// → NaN

a = [x];
// → [NaN]

a.includes(x);
// → true
```

```
x = 0 / 0;
// → NaN

a = [x];
// → [NaN]

a.includes(x);
// → true

a.indexOf(x);
```

```
x = 0 / 0;
// → NaN

a = [x];
// → [NaN]

a.includes(x);
// → true

a.indexOf(x);
// → -1
```

```
// a.indexOf(x)
a.indexOf(x)
```

```
// a.indexOf(x)
a.indexOf(x)
 → a.findIndex(v => v === x)
```

```
// a.indexOf(x)
a.indexOf(x)
 → a.findIndex(v => v === x)
 → -1
```

```
// a.indexOf(x)
a.indexOf(x)
 → a.findIndex(v => v === x)
 → -1

// a.includes(x)
a.includes(x)
 → a.find(v =>
            v === x ||
            (isNaN(v) && isNaN(x)))
```

```
// a.indexOf(x)
a.indexOf(x)
 → a.findIndex(v => v === x)
 → -1

// a.includes(x)
a.includes(x)
 → a.find(v =>
            v === x ||
            (isNaN(v) && isNaN(x)))
 → true
```

```
a = [
  "1",
  "2",
  "3"
];
// → ["1", "2", "3"]

a.map(parseInt);
```

```
a = [
  "1",
  "2",
  "3"
];
// → ["1", "2", "3"]

a.map(parseInt);
// → [1, NaN, NaN]
```

```
// a.map(parseInt)
["1","2","3"].map(parseInt)
```

```
// a.map(parseInt)
["1","2","3"].map(parseInt)
 → ["1","2","3"].map((value, index) => parseInt(value, index))
```

```
// a.map(parseInt)
["1","2","3"].map(parseInt)
 → ["1","2","3"].map((value, index) => parseInt(value, index))
 → [parseInt("1", 0), parseInt("2", 1), parseInt("3", 2)]
```

```
// a.map(parseInt)
["1","2","3"].map(parseInt)
 → ["1","2","3"].map((value, index) => parseInt(value, index))
 → [parseInt("1", 0), parseInt("2", 1), parseInt("3", 2)]
 → [1, parseInt("2", 1), parseInt("3", 2)]
```

```
// a.map(parseInt)
["1","2","3"].map(parseInt)
 → ["1","2","3"].map((value, index) => parseInt(value, index))
 → [parseInt("1", 0), parseInt("2", 1), parseInt("3", 2)]
 → [1, parseInt("2", 1), parseInt("3", 2)]
 → [1, NaN, parseInt("3", 2)]
```

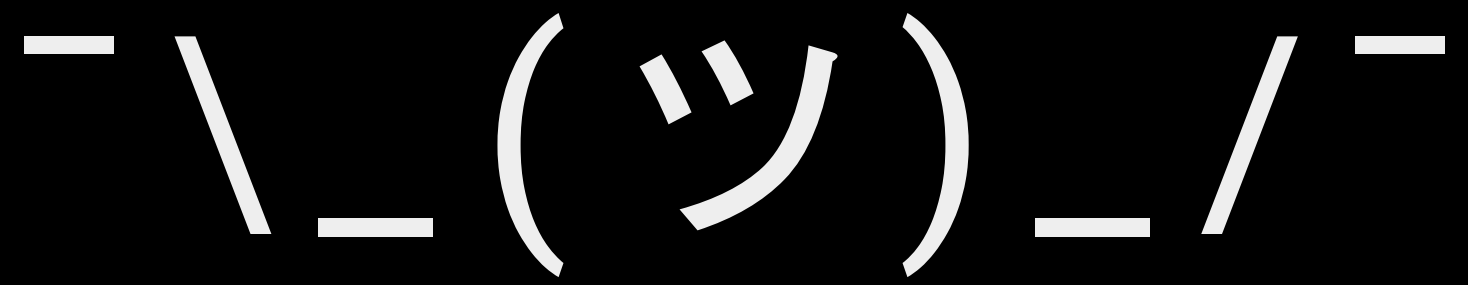
```
// a.map(parseInt)
["1","2","3"].map(parseInt)
 → ["1","2","3"].map((value, index) => parseInt(value, index))
 → [parseInt("1", 0), parseInt("2", 1), parseInt("3", 2)]
 → [1, parseInt("2", 1), parseInt("3", 2)]
 → [1, NaN, parseInt("3", 2)]
 → [1, NaN, parseInt("", 2)]
```

```
// a.map(parseInt)
["1","2","3"].map(parseInt)
 → ["1","2","3"].map((value, index) => parseInt(value, index))
 → [parseInt("1", 0), parseInt("2", 1), parseInt("3", 2)]
 → [1, parseInt("2", 1), parseInt("3", 2)]
 → [1, NaN, parseInt("3", 2)]
 → [1, NaN, parseInt("", 2)]
 → [1, NaN, NaN]
```

```
Array(25).fill(0/0).map(parseInt).join(" ");
```

@bmeurer

```
Array(25).fill(0/0).map(parseInt).join(" ");
// → "NaN NaN NaN NaN NaN NaN NaN NaN
//     NaN NaN NaN NaN NaN NaN NaN NaN
//     NaN NaN NaN NaN NaN NaN NaN NaN 13511"
```



@bmeurer

Thank you!

@bmeurer

@v8js