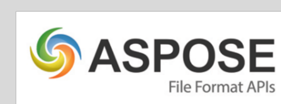
NDC { Sydney }



@megganeturner

When Your Code Does a Number on You

Navigating Numbers in Javascript

@megganeturner

Section 1:

What's in a ~~name~~ number?

@megganeturner

- count
- measure
- label
- identify

Classification

- Natural: 1, 2, 3, 4…
- Integer: -2, -1, 0, 1, 2…
- Rational: ½, ⅓, ¼…
- Irrational: π, √2…
- Real: 1, ½, 0.7, π, √2…

- Complex
- Transcendental
- Imaginary
- Infinity (& -infinity)
- Infinitesimals
- Surreal…

¯_(ツ)_/¯

Representation

8

八

आठ

@megganeturner

- 1234567
- 0b100101101011010000111
- 0o4553207
- 0x12D687
- 1.234567e6

mutilatium
A girl's got to have her standards.

IEEE-754

IEEE Standard for Floating-Point Arithmetic

Or why

`0.1 + 0.2 !== 0.3`

@megganeturner

- Specifies the implementation of floating-point arithmetic
- Allows us to represent real numbers as an approximation, to support a trade off between *range* & *precision*

Range:

-9007199254740991 – +9007199254740991

Precision:

17 decimal places (e.g. 0.30000000000000004)

01001000 01100101
01101100 01101100
01101111 00101100
00100000 01000100
01000100 01000100
00100001

"Hello, DDD!"

@megganeturner

But we can't have *decimal* points in binary

Floating-point arithmetic tries to account for this

1234567

01000001001100101101011010000111
00000000000000000000000000000000

64 bits

1 bit	11 bits	52 bits
0	10000010011	0010110101101000011100000000000000000000000000000000
Sign	Exponent	Significand / Mantissa

@megganeturner

```
> 0.1 + 0.2
<· 0.30000000000000004
```

@megganeturner

@megganeturner

Integers

-2, -1, 0, 1, 2

Fractions

½, ⅓, ¼, ⅕…

@megganeturner

$$\frac{1}{3} = 0.333\ldots$$

$$0.33 + 0.33 + 0.33 \mathrel{!==} 1.0$$

@megganeturner

0.1 (decimal)

0.000110011001100110011001100
1100110011001100110011001101
(binary)

0.1000000000000000555 (decimal)

@megganeturner

How big a problem is this really?

```
> 0.1 + 0.2
<· 0.30000000000000004
```

@megganeturner

0.0000000000000004cm

= 0.000000003 times as long as a Glucose Molecule

@megganeturner

0.00000000000000004km

= 0.00000000000000000001 times as long as The Distance from Earth to the Moon

@megganeturner

0.00000000000000004ly

= 38.38cm, or about the height of a standard bowling pin 🎳

@megganeturner

in most cases, this degree of accuracy is not going to be that important

Section 2:

Size really does matter

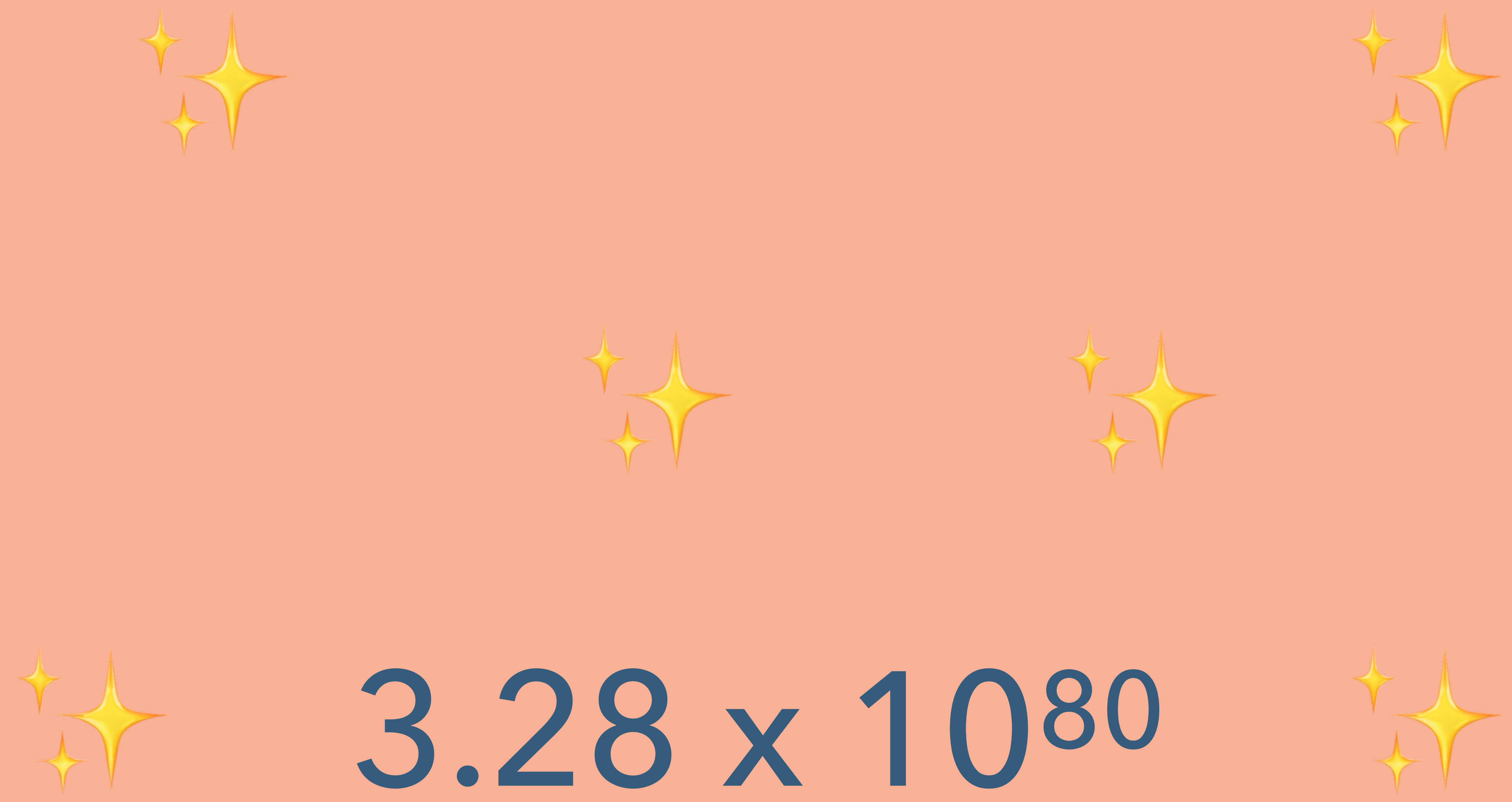
A trade off between *range* and *precision*

```
> 1.7e308
<· 1.7e+308
```

```
> 1.8e308
<· Infinity
```

```
> Number.MAX_VALUE
<· 1.7976931348623157e+308
```

```
> Number.MIN_VALUE
<· 5e-324
```



3.28×10^{80}

@megganeturner

```
> Number.MAX_SAFE_INTEGER
<· 9007199254740991
```

```
> Number.MIN_SAFE_INTEGER
<· -9007199254740991
```

(or ~9 quadrillion)

@megganeturner

```
> Number.MAX_SAFE_INTEGER
<· 9007199254740991

> Number.MAX_SAFE_INTEGER + 1
<· 9007199254740992

> Number.MAX_SAFE_INTEGER + 2
<· 9007199254740992

> Number.MAX_SAFE_INTEGER + 3
<· 9007199254740994

> Number.MAX_SAFE_INTEGER + 4
<· 9007199254740996

> Number.MAX_SAFE_INTEGER + 5
<· 9007199254740996

> Number.MAX_SAFE_INTEGER + 6
<· 9007199254740996
```

@megganeturner

```
<· ▼ (2) [{…}, {…}] ℹ
    ▼ 0:
        id: 110160323014044160
        name: "Beyoncé 👸🏽"
      ▶ __proto__: Object
    ▼ 1:
        id: 110160323014044160
        name: "Jay Z 🤴🏾"
      ▶ __proto__: Object
      length: 2
    ▶ __proto__: Array(0)
```

@megganeturner

```
(2) [{…}, {…}]
▼ 0:
    id: 1101603230140441 60
    name: "Beyoncé 👸🏽"
  ▶ __proto__: Object
▼ 1:
    id: 1101603230140441 60
    name: "Jay Z 🤴🏾"
  ▶ __proto__: Object
  length: 2
▶ __proto__: Array(0)
```

@megganeturner

```
> parseInt(110160323014044167)
  110160323014044160
> parseInt(110160323014044168)
  110160323014044160
```

@megganeturner

```
▼ (2) [{…}, {…}]
  ▼ 0:
      id: "c71193c8-8b80-4345-8102-41b5567fa7f7"
      name: "Beyoncé 👸🏽"
    ▶ __proto__: Object
  ▼ 1:
      id: "d9491461-ba2a-4825-a7ba-0ad869892317"
      name: "Jay Z 🤴🏾"
    ▶ __proto__: Object
    length: 2
  ▶ __proto__: Array(0)
```

solution: use UUIDs

@megganeturner

@megganeturner

Expecting Numbers?

- check them to make sure they're not larger than `MAX_SAFE_INTEGER`
- get them passed through as strings

This just in

Meggan @ DDD Sydney ✨ @megganeturner · Aug 13

Extremely rude of someone to advise me of a new number primitive in JS FOUR DAYS before I am supposed to talk about numbers in JS at @dddsydney

BigInt more like BigInterruption to my life 😰

(It's still very cool that this has shipped in V8 though! 🎉)

tc39/proposal-bigint

Arbitrary precision integers in JavaScript. Contribute to proposal-bigint development by creating an account on Github.

github.com

@megganeturner

BigInt: arbitrary-precision integers in JavaScript

By Mathias Bynens
V8 JavaScript whisperer

`BigInt`s are a new numeric primitive in JavaScript that can represent integers with arbitrary precision. With `BigInt`s, you can safely store and operate on large integers even beyond the safe integer limit for `Number`s. This article walks through some use cases and explains the new functionality in Chrome 67 by comparing `BigInt`s to `Number`s in JavaScript.

Use cases

Arbitrary-precision integers unlock lots of new use cases for JavaScript.

`BigInt`s make it possible to correctly perform integer arithmetic without overflowing. That by itself enables countless new possibilities. Mathematical operations on large numbers are commonly used in financial technology, for example.

Large integer IDs and high-accuracy timestamps cannot safely be represented as `Number`s in JavaScript. This often leads to real-world bugs, and causes JavaScript developers to represent them as strings instead. With `BigInt`, this data can now be represented as numeric values.

@megganeturner

`BigInt` could form the basis of an eventual `BigDecimal` implementation. This would be useful to represent sums of money with decimal precision, and to accurately operate on them (a.k.a. the `0.10 + 0.20 !== 0.30` problem).

@megganeturner

```
> BigInt(100)
<· 100n
```

```
> BigInt(100.5)
✖ ▶Uncaught RangeError: The          VM280384:1
number 100.5 is not a safe integer and
thus cannot be converted to a BigInt
    at BigInt (<anonymous>)
    at <anonymous>:1:1
```

```
> 6n + 3
✖ ▶Uncaught TypeError: Cannot mix VM98:1
BigInt and other types, use explicit
conversions
    at <anonymous>:1:4
```

@megganeturner

```
> Number.MAX_SAFE_INTEGER
<· 9007199254740991
> Number.MAX_SAFE_INTEGER + 1
<· 9007199254740992
> Number.MAX_SAFE_INTEGER + 2
<· 9007199254740992
> Number.MAX_SAFE_INTEGER + 3
<· 9007199254740994
> Number.MAX_SAFE_INTEGER + 4
<· 9007199254740996
> Number.MAX_SAFE_INTEGER + 5
<· 9007199254740996
> Number.MAX_SAFE_INTEGER + 6
<· 9007199254740996
```

@megganeturner

```
> var bigNum =
  BigInt(Number.MAX_SAFE_INTEGER)
```

```
> bigNum + 1n
<· 9007199254740992n
```

```
> bigNum + 2n
<· 9007199254740993n
```

```
> bigNum + 3n
<· 9007199254740994n
```

@megganeturner

```
> bigNum + 4n
<· 9007199254740995n
```

```
> bigNum + 5n
<· 9007199254740996n
```

```
> bigNum + 6n
<· 9007199254740997n
```

@megganeturner

Caveats

- Integers only
- V8 (Chrome) only
- Very new! Minimal documentation

Section 3:

Preaching to the converted

@megganeturner

Number.toString()

```
> 100..toString()
<· "100"
```

```
> 100 .toString()
<· "100"
```

```
> 100.0.toString()
<· "100"
```

Number.toString(base)

```
> 100..toString(2)
< "1100100"
```

```
> 100..toString(36)
< "2s"
```

```
> 100..toString(37)
✖ ▶Uncaught RangeError:                    VM2020:1
toString() radix argument must be between
2 and 36
    at Number.toString (<anonymous>)
    at <anonymous>:1:6
```

Number(string)

```
> Number('100')
<· 100
```

```
> Number('asd100')
<· NaN
```

@megganeturner

+string

```
> +'100'
<· 100

> +'asd100'
<· NaN
```

@megganeturner

-string

```
> -'-100'
<· 100
```

```
> -'100'
<· -100
```

@megganeturner

parseInt(string)

```
> parseInt('100')
<· 100
```

```
> parseInt('100.5')
<· 100
```

```
> parseInt('100asd')
<· 100
```

```
> parseInt('asd100')
<· NaN
```

@megganeturner

parseInt(string, radix)

```
> parseInt('11111111', 2)
<· 255
```

```
> parseInt('0xff', 16)
<· 255
```

parseFloat(string)

```
> parseFloat('100.5')
<· 100.5
```

```
> parseFloat('100')
<· 100
```

```
> parseFloat('asd100')
<· NaN
```

```
> parseFloat('100asd')
<· 100
```

@megganeturner

Number Object (Properties)

```
Number.MAX_SAFE_INTEGER

Number.MAX_VALUE

Number.MIN_SAFE_INTEGER

Number.MIN_VALUE
```

Number Object (Properties)

```
> Number.EPSILON
<· 2.220446049250313e-16
```

```
> Number.NaN
<· NaN
```

```
> Number.POSITIVE_INFINITY
<· Infinity
```

```
> Number.NEGATIVE_INFINITY
<· -Infinity
```

@megganeturner

Number Object (Methods)

```
> NaN === NaN
<· false
```

```
> Number.isNaN(NaN)
<· true
```

@megganeturner

Number Object (Methods)

```
> Number.isFinite(1)
<· true
```

```
> Number.isFinite(Infinity)
<· false
```

@megganeturner

Number Object (Methods)

```
> Number.isSafeInteger(1)
<· true
```

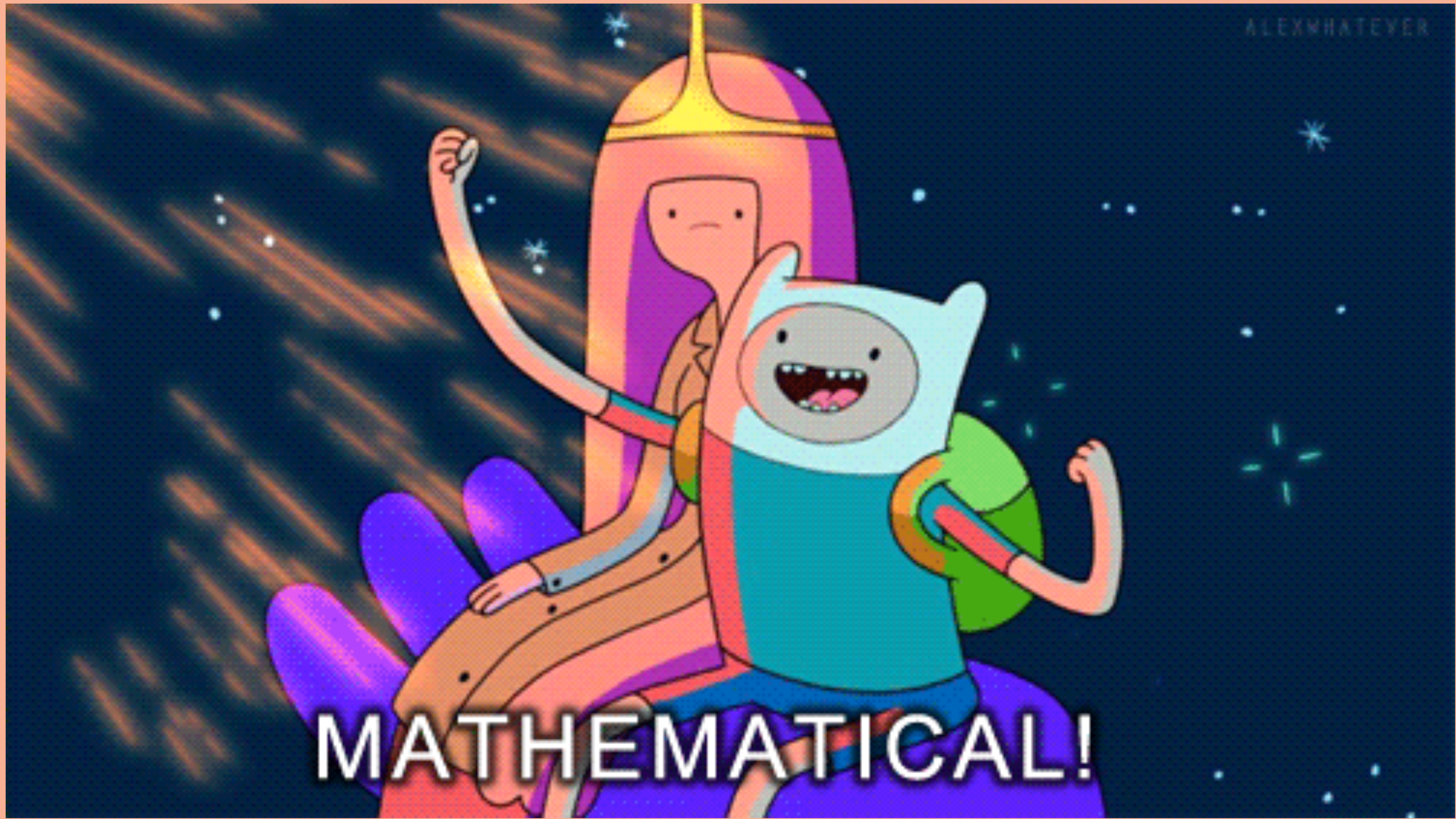
```
> Number.isSafeInteger(Number.MAX_SAFE_INTEGER + 1)
<· false
```

Number Object (Methods)

```
Number.parseFloat()

Number.parseInt()
```

@megganeturner


MATHEMATICAL!

Math Object (Properties)

```
Math.E
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
Math.PI
Math.SQRT1_2
Math.SQRT2
```

Math Object (Methods)

```
Math.abs(x)
Math.acos(x)
Math.acosh(x)
Math.asin(x)
Math.asinh(x)
Math.atan(x)
Math.atanh(x)
Math.atan2(y, x)
Math.cbrt(x)
Math.ceil(x)
Math.clz32(x)
Math.cos(x)
Math.cosh(x)
Math.exp(x)
Math.expm1(x)
Math.floor(x)
Math.fround(x)
Math.hypot([x[, y[, …]]])
Math.imul(x, y)
Math.log(x)
Math.log1p(x)
Math.log10(x)
Math.log2(x)
Math.max([x[, y[, …]]])
Math.min([x[, y[, …]]])
Math.pow(x, y)
Math.random()
Math.round(x)
Math.sign(x)
Math.sin(x)
Math.sinh(x)
Math.sqrt(x)
Math.tan(x)
Math.tanh(x)
Math.trunc(x)
```

Math Object (Methods)

Math.abs(x)

Math.acos(x)

Math.acosh(x)

Math.asin(x)

Math.asinh(x)

Math.atan(x)

Math.atanh(x)

Math.atan2(y, x)

Math.cbrt(x)

Math.ceil(x)

Math.clz32(x)

Math.cos(x)

Math.cosh(x)

Math.exp(x)

Math.expm1(x)

Math.floor(x)

Math.fround(x)

Math.hypot([x[, y[, …]]])

Math.imul(x, y)

Math.log(x)

Math.log1p(x)

Math.log10(x)

Math.log2(x)

Math.max([x[, y[, …]]])

Math.min([x[, y[, …]]])

Math.pow(x, y)

Math.random()

Math.round(x)

Math.sign(x)

Math.sin(x)

Math.sinh(x)

Math.sqrt(x)

Math.tan(x)

Math.tanh(x)

Math.trunc(x)

@megganeturner

Math Object (Methods)

```
> Math.abs(1)
<· 1
```

```
> Math.abs(-1)
<· 1
```

@megganeturner

Math Object (Methods)

```
> Math.ceil(1.2)
<· 2
```

```
> Math.floor(1.2)
<· 1
```

```
> Math.round(1.2)
<· 1
```

Math Object (Methods)

```
> Math.pow(2,3)
<· 8
```

```
> Math.random()
<· 0.4862522156481832
```

```
> Math.sqrt(4)
<· 2
```

Context is everything

gifak-net
h_2
h_1
b
a
S=a *h1
S=b *h2

Thank you!