* In JS, numerele sunt mereu decimale. Deci:

*console*.log(10 === 10.0)

true

* JS foloseste binary format pentru a stoca numerele. Deci, numerele sunt reprezentate intern doar cu 0 si 1. Alte limbaje folosest sistemul hexadecimal, adica numerele sunt reprezentate prin cifre de la 0 la 9.
* In sistemul hexadecimal, reprezentarea numerelor flotante sau a fractiilor e mult mai usoara, insa in cel binar de dificila
* Iata de ce, de ex, 3/10 va fi ceva de genul 0.33333333335
* Numerele mari pot fi scrise si asa: 2\_156\_145\_100, oricum \_ va fi ignorat. Totusi, nu putem converti in numere un string ce contine \_, gen ‘1\_000’ nu poate fi convertit in Number!

**Conversion**

1. Number(‘3’)
2. +’3’

*console*.log(+'0.3')

JS, cand vede + in fata la un string, incearca sa il converteasca in Number

**Number methods**

* **parseInt(‘’)** – extrage numarul gasit din string. String poate contine si alte caractere decat cifre, important sa inceapa cu numar:

*console*.log(Number.parseInt('3px'))

3

E bun cand primim valori de la CSS, gen 30px, 20em

* **parseFloat(‘’)** – extrage numarul flotant din string, chiar daca mai sunt si alte caarctere
* **isNaN(expression)** – returneaza true daca ave NaN sau false daca nu

*console*.log(Number.isNaN(Number('10x')))

true

* **isFinite(expression)** – verifica daca nu avem NaN sau Infinity.Finite sunt: NaN, Infinity. **Mai bine folosim isFinite decat isNaN**

De ex, 10/0 nu ne ofera NaN, ci infinity

*console*.log(10/0)

Infinity

Si problema e ca isNaN va returna false

*console*.log(Number.isNaN(10/0))

false

isFinite rezolva problema asta si ia in calcul si Infinity

**Math**

* **sqrt() -** radical
* **max(nr1, nr2, ...,)**
* **min(nr1, nr2, ...)**
* **random()** – nr intre 0 si 1
* **trunc()** – scoate partea zecimala
* **round()** – aproximeaza ca in matem, pana la intreg
* **ceil()** – aproximeaza in sus, la intreg
* **floor()** – aproximeaza in jos, la intreg
* **number.toFixed(nr)** – nr arata cate cifre dupa virgula sa se returneze, Returneaza un string!!! Face aproximarea
* *console*.log(25.1216.toFixed(2))

25.12

**BigInt**

* Este un nou tip de date primitive
* Contine numere foarte mari
* Pentru a transforma un numar in BigInt punem ‘n’ la urma:

*console*.log(46146545416145151320321650320202n)

* Putem face operatii cu bigint ca orice numar:

*console*.log(5100n + 8500n)

* Nu putem amestecta number cu bigint:

*console*.log(46146545416145151320321650320202n \* 25)

TypeError: Cannot mix BigInt and other types, use explicit conversions

* Putem folosi <, > etc.

*console*.log(46146545416145151320321650320202n > 25)

* Math nu functioneaza cu BigInt!
* BigInt niciodata nu accepta si nu returneaza numere cu virgula!