number.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML
    =typeof Infinity;
</script>
</html>
</body>
```

Infinity is a number
number

Hexadecimal:-

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x=0xFF;
document.getElementById("demo").innerHTML
    ="0xFF"+x;
</script>
</body>
</html>
```

⇒ 0xFF=25

→ Never write a number with leading zero like 07.
→ Some javascript versions interpret numbers as octal if they are written with a leading zero.
→ By default, javascript displays numbers as base 10 decimals.
→ But you can use the tostring() method to output numbers from base 2 to base 36.
→ Hexadecimal is base 16, Decimal is base 10.
Octal is base 8. Binary is 2.

```
let <!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
```

```
let myNumber=32;
document.getElementById("demo").innerHTML
="Decimal 32= "+"<br><br>"+
"Hexabigensigmal (base 36):"+myNumber.toString
(36)+"<br>"+"Duobigesimal (base 32):"
+myNumber.toString(32)+"<br>"+"Hexadeci-
mal (base 16):"+myNumber.toString(16)+"<br>"
Duodecimal (base 12) +myNumber.toString(12)
+"<br>"+"Decimal (base 10):-"+myNu-
mber.toString(10)+"<br>"+"Octal (base 8):"+
myNumber.toString(8)+"<br>"+"Binary (base2):
"+myNumber.toString(2);
</script>
</html>
</body></html>
```

Javascript Numbers as objects:-

```
let x="123";
```

numbers also created with the keywod new.

```
let y=new Number(123);
```

When using the operator(==), x&y are equal.

```
let x=500;
let y=new Number(500);
```

When using the (===) operator, x&y are not equal.

```
let x=500;
let y=new Number(500);
```

→ `let x= newNumber(500);`

`let y= new Number(500);`

`"x===y  let x= new Number(500);`

Big Int:-

→ these are used to store big integer values that are too big to be represented by a normal javascript number.

→ Javascript integers are only accurate upto 15-digits.

→ let x = 999999999999999; (15)
let y = 9999999999999999;

→ In javascript, all numbers are stored in a 64-bit floating-point format.

→ with this standard, large integer cannot be exactly represented and will be rounded

→ Because of this, Javascript can only safely represent integers:-

up to 9007199254740991 ($2^{53}-1$)

and

down to -9007199254740991 $-(2^{53}-1)$

How to create BigInt:-

To create a BigInt, append n to the end of an integer or call BigInt():-

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x = 999999999999999;
let y = BigInt("999999999999999");
document.getElementById("demo").innerHTML
    = x + "<br>" + y;
</script>
</body>
</html>
```

```
let x = 1234567890123456789012345n;
let y = BigInt(1234567890123456789012345)
```

BigInt:- A new javascript Datatype:-

The javascript type of a BigInt is "bigint".

```
let x = BigInt(999999999999999);
let type = typeof x;
```

→ BigInt is the total number of supported data type in javascript is 8:-

1. String
2. Number
3. BigInt
4. Boolean
5. undefined

let x = 9007199254740995n;
let y = 9007199254740995n;
let z = x * y;

→ Arithematic between a BigInt and a Number is not allowed (type conversion lose information)
→ unsigned right shift (>>>) can not be done on a BigInt (it does not have a fixed width)

BigInt Decimals:-

BigInt can not have decimals

```
let x = 5n;
let y = x / 2;
//Error: Cannot mix BigInt and other types,
        Use explicit conversion.
let x = 5n;
let y = Number(x) / 2;
```

BigInt Hex, Octal & Binary:-

```
let hex = 0x2000000000003n;
let oct = 0o400000000000003n;
let bin = 0b1000000000000000000000000000011n;
```

Precision Vulnerability:-

Rounding can compromise program security.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x = 900719925474099n === 900719925474099n;
document.getElementById("demo").innerHTML
    = x;
</script></body>
</html>
```

Minimum and maximum safe integers:-

MAX_SAFE_INTEGER

MIN_SAFE_INTEGER.

let x = Number.MAX_SAFE_INTEGER;

let x = Number.MIN_SAFE_INTEGER;

New Number methods:-

added 2 new methods to the number object.

Number.isInteger()

Number.isSafeInteger()

Number.isInteger():-

Number.isInteger() method returns true if the argument is an integer

Number.isInteger(10);

Number.isInteger(10.5);

Number.isSafeInteger():-

→ A safe integer is an integer that can be exactly represented as a double precision number.

→ The Number.isSafeInteger() method returns true if the argument is a safe integer.

Number.isSafeInteger(10);

Number.isSafeInteger(1234567890123456789);

Javascript Number methods:-

These number methods can be used on all JavaScript numbers:-

toString() → returns a number as a string

toExponential() → returns a number written in exponential notation.

toFixed() → returns a number written

toPrecision() returns a number written with a specified length

valueOf() returns a number as a number.

→ The tostring() method returns a number as a string.

→ All number methods can be used on any type of numbers (literals, variables ore expressions):-

```
let x=123;
x.toString();
(123).toString();
(100+23).toString();
```

The Exponetial method:-

→ toExponential() returns, a string with a number rounded and written using exponential notation.

→ A parameter defines the number of characters behind the decimal point:-

```
let x=9.656;
x.toExponetial(2);
x.toExponetial(4);
x.toExponetial(6);
```

toFixed():-

toFixed() returns a string, with the number written with a specified number of decimals.

```
let x=9.656;
x.toFixed(0);
x.toFixed(2);
x.toFixed(4);
x.toFixed(6);
```

→ toFixed(2) is perfect for working with money

toprecision Method():-

→ toprecision() returns a string, with a number written with a specified ~~no~~ length.

```
let x=9.656;
x.toprecision();
x.toprecision(2);
x.toprecision(4);
```

valueOf() returns a number as a ...

```
let x = 123;
x.valueOf();
(123).valueOf();
(100+23).valueOf();
```

→ In JavaScript, a number can be primitive value (typeof = number) or an object (typeof = object). The valueOf() method is used internally in JavaScript to convert number objects to primitive values.
→ There is no reason to use it your code.
→ All JavaScript datatypes have a valueOf() & toString() method.

Converting variables to Numbers:-

3-JavaScript methods that can be used to convert a variable number.

Number() → returns a number converted from its argument.

parseFloat() → parses its argument and returns a floating point number.

parseInt() → parses its argument and returns a whole number.

Number:-

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML
= Number(true)+"<br>"+
Number(false)+"<br>"+
Number("10")+"<br>"+
```

```
Number("10.33") + "<br>" +
Number("10,33") + "<br>" +
Number("10 33") + "<br>" +
Number("John");
</script>
</body>
</html>
```

→ If a number cannot be converted, NaN (Not a Number) is returned.

The Number() Method used on dates:-

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x = new Date("1970-01-01");
document.getElementById("demo").innerHTML
    = Number(x);
</script>
</body>
</html>
```

→ The Number() method can convert a date to a number of milliseconds 1.1.1970.

→ The number of milli seconds between 1970-01-02 & 1970-01-01 is 86400000:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x = new Date("1970-01-02");
document.getElementById("demo").inn
```

</body>
</html>

⇒ 86400000.

(23)
```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x = new Date("2017-09-30");
document.getElementById("demo").innerHTML=
Number(x);
</script>
</body>
</html>
```

⇒ 1506729600000

The parseInt() Method:-

parseInt() parses a string and returns a whole number. spaces are allowed. only the first number is returned:-

```
parseInt("-10");
parseInt("-10.33");
parseInt("10");
parseInt("10.33");
parseInt("10 20 30");
parseInt("10 years");
parseInt("years 10");
```

→ If the number cannot be converted, it will return NaN.

The parseFloat():-

parseFloat() parses a string and returns a number. spaces are allowed. only the first

parseFloat("10 20 30");
parseFloat("10 years");
parseFloat("years 10");

→ If the number cannot be converted NaN is returned.

Number object Methods:-

These object methods belong the Number object:-

Number.isInteger()
Returns true if the argument is an integer.

Number.isSafeInteger()
Returns true if the argument is safe integer.

Number.parseFloat()
Converts a string to a number.

Number.parseInt()
Converts a string to a whole number.

→ The number methods above belong to the JavaScript Number object.
→ These methods can only be accessed like Number.isInteger().
→ using x.isInteger() where x is a variable, will result in an error.

TYPEError x.isInteger is not a function.

The Number.isInteger method:-

returns true if the argument is an Integer.

Number.isInteger(10);
Number.isInteger(10.5);

The Number.isSafeInteger() method:-

→ A safe integer is an integer that can be exactly represented as a double precision number.
→ The Number.isSafeInteger() method returns true

parseFloat("10");
parseFloat("10.33");
parseFloat("10 20 30");
parseFloat("10 years");
parseFloat("years 10");
→ If the number cannot be converted, Nan is returned.

Number object Methods:-

these object methods belong the Number object:-

Number.isInteger()
Returns true if the argument is an integer.

Number.isSafeInteger()
Returns true if the argument is safe integer.

Number.parseFloat()
Converts a string to a number.

Number.parseInt()
Converts a string to a whole number.

→ the number methods above belong to the JavaScript Number object.
→ these methods can only be accessed like Number.isInteger().
→ using x.isInteger() where x is a variable, will result in an error.
TYPEError x.isInteger is not a function.

The Number.isInteger Method:-

returns true if the argument is an integer.

Number.isInteger(10);
Number.isInteger(10.5);

The Number.isSafeInteger() Method:-

→ A safe integer is an integer that can be exactly represented as a double precision number.
→ The Number.isSafeInteger() method returns true