New number oparator

Exponentiation

```
console.log(2**4)
// 16
```

New Method

Includes

The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

```
const array1 = [1, 2, 3];

console.log(array1.includes(2));
// expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// expected output: true

console.log(pets.includes('at'));
// expected output: false
```

Examples

```
[1, 2, 3].includes(2)      // true
[1, 2, 3].includes(4)      // false
[1, 2, 3].includes(3, 3)   // false
[1, 2, 3].includes(3, -1)  // true
[1, 2, NaN].includes(NaN)  // true
```

PadStart

The `padStart()` method pads the current string with another string (multiple times, if needed) until the resulting string reaches the given length. The padding is applied from the start of the current string.

```
const str1 = '5';

console.log(str1.padStart(2, '0'));
// expected output: "05"

const fullNumber = '2034399002125581';
const last4Digits = fullNumber.slice(-4);
const maskedNumber = last4Digits.padStart(fullNumber.length, '*');

console.log(maskedNumber);
// expected output: "************5581"
```

PadEnd

The padEnd() method pads the current string with a given string (repeated, if needed) so that the resulting string reaches a given length. The padding is applied from the end of the current string.

```
const str1 = 'Breaded Mushrooms';

console.log(str1.padEnd(25, '.'));
// expected output: "Breaded Mushrooms........"

const str2 = '200';

console.log(str2.padEnd(5));
// expected output: "200  "
```

Array forEach

The forEach() method executes a provided function once for each array element.

```
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

Entries

The Object.entries() method returns an array of a given object's own enumerable string-keyed property [key, value] pairs, in the same order as that provided by a for...in loop. (The only important difference is that a for...in loop enumerates properties in the prototype chain as well).

```
const obj = { foo: 'bar', baz: 42 };
console.log(Object.entries(obj)); // [ ['foo', 'bar'], ['baz', 42] ]

// array like object
const obj = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.entries(obj)); // [ ['0', 'a'], ['1', 'b'], ['2', 'c'] ]
```

async function

An async function is a function declared with the async keyword. Async functions are instances of the AsyncFunction constructor, and the await keyword is permitted within them. The async and await keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

```
const fetchData = async(url) => {
   let data = { name : 'hesam' , family : 'mousavi'};
   if(true) {
      return data;
   } else {
      throw new Error('error URL');
   }
}


const saveDataToDB = async(data) => {
   if(true) {
      return true;
   } else {
      throw new Error('error DB')
   }
}


 fetchData('roocket.ir')
    .then((data) => saveDataToDB(data))
    .then((status) => console.log(status))
    .catch((err) => console.log(err));
```

Await

```
const fetchData = async(url) => {
   let data = { name : 'hesam' , family : 'mousavi'};
   if(true) {
      return data;
   } else {
      throw new Error('error URL');
   }
}


const saveDataToDB = async(data) => {
   if(true) {
      return true;
   } else {
      throw new Error('error DB')
   }
}

let executeFetchData = async() => {
   try {
      let data = await fetchData('roocket.ir');
      let statusDB =  await saveDataToDB(data);
      console.log(statusDB);
   } catch(err) {
      console.log(err);
   }
}

executeFetchData();
```