1. В массиве найти максимальное число

```
const numbers = [1, 45, 66, 2, 4, 6, -2, -10, -100, 100]
let minValue = numbers[0]
for (let i = 0; i < numbers.length; i++) {
   if (numbers[i] < minValue) {
      minValue = numbers[i]
   }
}
console.log(minValue)
```

```
const numbers = [1, 45, 66, 2, 4, 6, -2, -10, -100, 100]
let maxValue = numbers[0]
for (let i = 0; i < numbers.length; i++) {
   if (numbers[i] > maxValue) {
      maxValue = numbers[i]
   }
}
console.log(maxValue)
```

2. Рекурсия: фибоначчи, факториал, сумма чисел от 0 до N

a) Факториал (теория)

```
 function factorial(n) {
    return n ? n * factorial(n - 1) : 1
 }

 alert(factorial(5)) // 120"
```

b) Fibonacci (теория)

```
function fib(n) {
 return n <= 1 ? n : fib(n - 1) + fib(n - 2);
}

alert( fib(3) ); // 2
alert( fib(7) ); // 13
// fib(77); // не запускаем, подвесит браузер"
```

c) Сумма n чисел (теория)

```
function sumTo(n) {
    if (n == 1) return 1
    return n + sumTo(n - 1)
 }

 alert(sumTo(100))
```

d) Рекурсия (теория)

3. Call, apply, bind

a) bind (не выполняет функцию, возвращает другую функцию с навсегда заданным контекстом):

```
function foo() {
      console.log(this.name);
  }

  let a = { name: 'Dima' };
  let b = { name: 'Viktor' };

  const bindedFooA = foo.bind(a);
  const bindedFooB = foo.bind(b);

  bindedFooA(); // 'Dima'
  bindedFooB(); // 'Viktor'

  bind* более сложный пример с параметрами:

  function foo(age, city) {
      console.log(`${this.name}, ${age}, ${city}`);
  }

  let a = { name: 'Dima' };
  let b = { name: 'Viktor' };

  const bindedFooA = foo.bind(a, 30);
  const bindedFooB = foo.bind(b, 18);

  bindedFooA('Tbilisi'); // Dima, 30, Tbilisi
  bindedFooB('Minsk'); // 'Viktor, 18, Minsk
```

b) apply\call (сразу выполняют функцию, разница двух функций в том, как передавать параметры... apply - array, call - comma (запятая))

```
function foo(age, city) {
   console.log(`this.name, ${age}, ${city}`)
}

let a = { name: 'Dima' }
let b = { name: 'Viktor' }

foo.apply(a, [31, 'Tbilisi'])
foo.call(b, 18, 'Minsk')
```

4. map, filter, reduce

a) map

map возвращает **НОВЫЙ** массив.
map нужен, чтобы из массива, в котором содержаться элементы в оригинальном виде, получить массив той же длины, который сожержит "новые" элементы, полученные на основе элементов старого массива:

```
  ['1', '2', '3'].map((el) => +el) // массив строк преобразовываем в массив чисел
  [18, 20, 12].map((age) => {
     if (age >= 18) {
        return { age: age, adult = true }
     } else {
        return { age, adult = false }
     }
  }) // из массива чисел получаем массив объектов, с полями: age и adult (взрослый): true\false в зависимости от того, возраст >= 18 или нет
```

b) filter

filter возвращает НОВЫЙ массив.
filter нужен, чтобы получить новый отфильтрованный массив, в котором будет меньше элементов, чем в исходном, потому что мы фильтруем исходный, убираем ненужное:

```
['Minsk', 'Moscow', '', '', 'London', ''].filter( (el) => el !== ''); // пропускам в результирующий массив не пустые строки
  [{age: 18, sex: 'f', name: 'Sveta'}, {age: 17, sex: 'f', name: 'Sashka'}, {age: 19, sex: 'm', name: 'Andrew'}].filter( (person) => {
        return person.age >= 18 && person.sex === 'f';
  }) // пропускаем на вечеринку только тех, кто девочка и кому 18+}
```

c) reducer

reducer пробегается по всему массиву и на выход выдаёт какое-то одно обобщённое значение. Это может быть как новый массив, так и простое значение примитив или объект:

```
['Minsk', 'Moscow', '', '', 'London', '']
    .reduce((acc, el) => {
       if (el !== '') acc++
       return acc
    }, 0) // подсчитываем, сколько у нас в массиве не пустых строк

    [
       { age: 18, sex: 'f', name: 'Sveta' },
       { age: 17, sex: 'f', name: 'Sashka' },
       { age: 19, sex: 'm', name: 'Andrew' }
    ].reduce((acc, person) => {
       if (person.age >= 18 && person.sex === 'f') {
          acc.push(person)
       }
       return acc
    }, []) // на выходе получаем новый массив, состоящий из людей, кто девочка и
кому 18+ (но лучше эту задачу решать с помощью filter, но и так можно)

    [1, 4, 6, 66, -12].reduce((acc, number) => {
       acc += number
       return acc
    }, 0) // подсчёт суммы всех чисел в массиве
```

5. Замыкание: counter

```
function makeCounter() {
    var currentCount = 1
    return function () {
       // (**)
       return currentCount++
    }
 }

 var counter = makeCounter() // (*)

 // каждый вызов увеличивает счётчик и возвращает результат
 alert(counter()) // 1
 alert(counter()) // 2
 alert(counter()) // 3

 // создать другой счётчик, он будет независим от первого
 var counter2 = makeCounter()
 alert(counter2()) // 1
```

6. Наследование, пример на class\extends

```
class Animal {
    constructor(name) {
       this.name = name
    }
    walk() {
       alert('I walk: ' + this.name)
    }
    eat() {
       alert('I can eat')
    }
 }

 class Rabbit extends Animal {
    walk() {
       super.walk()
       alert('...and jump!')
    }
 }

 var rabbit = new Rabbit('Bunny')
 rabbit.walk()
 rabbit.eat()
```

7. Промисификация, setInterval, setTimeout

```
doItAfter(2).then(() => console.log())

 function doItAfter(seconds) {
    let promise = new Promise((resolve, reject) => {
       setInterval(() => {
          resolve()
       }, seconds * 1000)
    })
    return promise
 }
```