**13. Условия**

// условие классическое

      const num = 50;

      if (num < 49) {

          console.log('error');

      } else if (num > 100) {

        console.log('много');

      } else {

        console.log('ok!');

      }

      //тернарный оператор     --- участвует три аргумента

      const num = 50;

      (num === 50) ? console.log('ok!') : console.log('error!');

      //switch

      const num = 50;

      switch (num) {

          case 49:

            console.log('error!');

            break;

          case 100:

            console.log('error!');

            break;

          case 50:

                console.log('ok!');

                break;

          default:

            console.log('Не в этот раз!');

      }

**14. Циклы**

  // 1. Пока

      let num = 50;

      while (num <= 55) {

          console.log(num);

          num++;

      }

      // 2. do делать что-то, пока while

      let num = 50;

      do {

          console.log(num);

          num++;

      }

      while (num <= 55)

      //3. for для

      let num = 50;

      for (let i = 0; i <= 5; i++ ) {

          console.log(num);

          num++;

      }

      // операторы break и continue

      for (let i = 0; i <= 10; i++ ) {

          if (i ===6) {

            break;   // прерываемся на i === 6

          }

          console.log(i);

      }

      for (let i = 0; i <= 10; i++ ) {

        if (i ===6) {

            continue;   // пропускаем i === 6 и продолжаем дальше

        }

        console.log(i);

    }

**16. Функции, стрелочные ф-ции (ES6)**

// 1. FUNCTION DECLARATION

function showFirstMesagge() {

    console.log('Hello world');

}

showFirstMesagge();

let num = 20;

function showFirstMesagge(text) {

    console.log(text);

    let num = 10;

}

showFirstMesagge('Hello world');

console.log(num);

// function calc(a, b) {

//     return a + b;

// }

// console.log(calc(4, 7));

// console.log(calc(5, 12));

// выходит num наружу

function ret() {

    let num = 50;

    return num; // выходит num наружу (с функции)

}

const actors = ret();

console.log(actors);

// Замыкание функций

let num2 = 20;

function showFirstMesagge(text) {

    console.log(text);

    console.log(num2); // ищет вышестоящее значение

}

showFirstMesagge('Hello world');

console.log(num2);

// 2. FUNCTION EXPRESSION

const log = function() {

    console.log('Hello world');

};

log();

// 3. СТРЕЛОЧНЫЕ ФУНКЦИИ

const log1 = () => {

    console.log('Hello world');

};

log1();

const calc = (a,b) => {

    return a + b;

};

console.log(calc(4, 7));

console.log(calc(5, 12));

**17. Методы и свойства строк и чисел**

// 17. Методы и свойства строк и чисел

const str = 'test';

console.log(str.length); // свойство строки

// методы строк

const strUpper = str.toUpperCase(); // метод строки

//console.log(str.toUpperCase());

console.log(str);

console.log(strUpper);

// метод поиск подстроки

const fruit = 'Some fruit';

console.log(fruit.indexOf('fruit')); // поиск подстроки

// метод slice()

const logg = 'Hello world';

console.log(logg. splice

(6, 11)); // вырезает кусочек строки с 6 по 11 индекс. (получаем world)

// метод substring()   тоже самое что и slice()

// метод substr()

console.log(logg.substr(6, 4)); // вырезает кусочек с 6 индекса (2 цифра сколько вырезать)

// Числа

const number = 12.2;

console.log(Math.round(number)); // округление (например до 12)

const test = '12.2px';

console.log(parseInt(test)); // метод превращает строку в целое число (12)

console.log(parseFloat(test)); //метод превращает строку в число (12.2)

19. Callback- функции

/19. Callback- функции

function learnJS(lang, callback) {

    console.log(`Я учу ${lang}`);

    callback();

}

function done() {

    console.log('Я прошел этот урок');

}

learnJS('JS', done);

learnJS('JS', function(){

    console.log('Я прошел этот урок');

});

20. Объекты, деструктуризация объектов (ES6)

// 20. Объекты, деструктуризация объектов (ES6)

const options = {

    name: 'text',

    widht: 1024,

    height: 1024,

    colors: {

        border: 'black',

        bg: 'red'

    },

    makeTest: function() {   // делаем метод в объекте

        console.log('test');

  }

};

options.makeTest();  // запускаем метод    круто!!!

//деструктуризация объектов (ES6)

const {border, bg} = options.colors;

console.log(border);

// узнать количество свойств в объекте

console.log(Object.keys(options).length);

// узнать свойство в объекте

//console.log(options.name);

// перебор свойств в объекте

for (let key in options) {

    if (typeof(options[key]) === 'object') {

        for (let i in options[key]) {

            console.log(`Свойство ${i} равно ${options[key][i]}`);

        }

    } else {

        console.log(`Свойство ${key} равно ${options[key]}`);

    }

}

**21. Массивы и псевдомассивы**

const arr = [1, 2, 3, 6, 8];

arr[99] = 7;

console.log(arr.length); // свойство length +1 !!!

console.log(arr);

// методы которые работают с концом массива

arr.pop(); // удаляет последнее элемент массива [1, 2, 3, 6]

arr.push(10); // добавляет в конец элемент массива [1, 2, 3, 6, 8, 10]

// перебор элементов в массиве

// метод forEach !!!

arr.forEach(function(item, index, arr){

    console.log(item);

});

for (let i = 0; i < arr.length; i++) {

    console.log(arr[i]);

}

for (let i of arr) {

    console.log(i);

}

// метод split('.')          из строки (через точку('.')) превращает в массив

const str3 = prompt("", "");

const products = str3.split('.');

console.log(products);

// метод join(";")   наоборот       из массиа (через (";")) превращает в строку

const str4 = prompt("", "");

const products1 = str4.split('.');

products1.sort();     // метод sort() сортирует строки

console.log(products1.join(";"));

// метод sort()

const arr2 = [21, 5, 10, 82, 9];

arr2.sort(compareNum);

console.log(arr2);

function compareNum(a, b) {  // с этой функцией метод sort сортирует числа по порядку.

    return a - b;

}

**22. Передача по ссылке или по значению, Spread  
оператор (ES6-ES9)**

**Клонирование объектов**

# // Object.assign()

const numbers = {

    a: 10,

    b: 12,

    c: {

        d: 2,

        e: 5

    }

};

const add = {

    f: 10,

    k: 2

};

const clone = Object.assign({}, numbers);

clone.a = 20;

clone.d = 10;

console.log(clone);

console.log(numbers);

# // Spread

// Оператор разворота Spread (...)

//массивы

const video = ['youtube', 'vimio', 'rutube'],

    blogs = ['wordpress', 'livejouurnal', 'blogger'],

    internet = [...video, ...blogs, 'vk', 'facebook'];

    console.log(internet);

function log(a, b, c) {

    console.log(a);

    console.log(b);

    console.log(c);

}

const num = [1, 5, 10];

log(...num);

const array = ['a', 'b'];

const newArray = [...array];

//объекты

const obj = {

    a: 1,

    b:5,

    c: {

        f: 15,

        g: 20

    }

};

const newObj = {...obj};

newObj.a = 10;

console.log(newObj);

console.log(obj);

# 23. Основы ООП, прототипно-ориентированное наследование

const soldier = {

    health: 400,

    armor: 100,

    sayHello: () => {

        console.log('Hello');

    }

};

// const jonh = {

//     health: 100

// };

//jonh.\_\_proto\_\_ = soldier; // прототип старый \_\_proto\_\_

//Object.setPrototypeOf(jonh, soldier); // прототип новый с помощью Object.setPrototypeOf

const jonh = Object.create(soldier); // создаем новый объект-прототип с помощью Object.create

jonh.health = 100;

console.log(jonh.health);

jonh.sayHello();

**28. Получение элементов со страницы**

const hearts = document.querySelectorAll('.heart'); // получаем массив из heart

console.log(hearts);

hearts.forEach(item => {     // можно использовать forEach  --- перебор hearts

    console.log(item);

});

const oneHeart = document.querySelector('.heart');    // выбираем первый элемент в массиве

console.log(oneHeart);                               // удобно использовать когда элемент уникален тоесть один)

**29. Действия с элементами на странице**

const box = document.querySelector('.box'),

    btns = document.querySelectorAll('button'),

    circles = document.querySelectorAll('.circle'),

    hearts = document.querySelectorAll('.heart'),

    wrapper = document.querySelector('.wrapper');

// box.style.backgroundColor = "blue"; //назначаем инлайн стили

// box.style.width = "500px";

box.style.cssText = 'background-color: blue; width: 500px'; // используем cssText

btns[1].style.borderRadius = '100%';

circles[0].style.backgroundColor = 'red';

//назначаем стили на все елементы

// есть несколько методов

// через Цикл   но его практически не используются

// for (let i = 0; i < hearts.length; i++) {

//     hearts[i].style.backgroundColor = 'green';

// }

//через forEach

hearts.forEach(item => {

    item.style.backgroundColor = 'green';

});

//создаем елемент внутри JavaScripta

const div = document.createElement('div'); // здесь ('div');можно любое значение, главное с буквы начало

//добавляем класс к элементу

div.classList.add('black');

//размещаем элемент на странице

//append --- в конец

//document.body.append(div);  // добавляем в конец body

wrapper.append(div);// добавляем в конец wrapper

// prepend --- в начало

//wrapper.prepend(div); // добавляем в начало wrapper

//before --- перед элементом

//hearts[0].before(div);

// after --- после элемента

//hearts[0].after(div);

// удаляем элемент со страницы

//circles[1].remove();

// заменяем элемент другим элементом

//hearts[0].replaceWith(circles[0]);

//всталяем текст в div

div.innerHTML = '<h1>hello world</h1>';

//div.textContent = 'hello world';

//вставляем HTML код

//перед элементом div

//div.insertAdjacentHTML('beforebegin', '<h2>Hello</h2>');

//в начало элемента div

//div.insertAdjacentHTML('afterbegin', '<h2>Hello</h2>');

//в конец элемента div

//div.insertAdjacentHTML('beforeend', '<h2>Hello</h2>');

//перед элементом div

//div.insertAdjacentHTML('afterend', '<h2>Hello</h2>');

**30. Практика**

**31. События и их обработчики**

const btns = document.querySelectorAll('button'),

    overlay = document.querySelector('.overlay');

//навешиваем обработчик события + объект события "event"

    // btn.addEventListener('click', (event) => {

    //     console.log(e.target);

    //     e.target.remove();

    //     //alert('click');

    // });

// удаляем обработчик события после 1 нажатия на кнопку

    //     let i = 0;

    const deleteElement = (event) => {

        console.log(event.target);

    //     i++;

    //     if (i == 1) {

    //         btn.removeEventListener('click', deleteElement);

    //     }

     };

    //    btn.addEventListener('click', deleteElement);

//всплытие событий   сначала срабатывает на вложенном элементе, потом  по иерархии вверх на родителе

//     let i = 0;

// const deleteElement = (event) => {

//     console.log(event.target);

//     i++;

//     if (i == 2) {

//         btn.removeEventListener('click', deleteElement);

//     }

// };

//     btn.addEventListener('click', deleteElement);

//     overlay.addEventListener('click', deleteElement);

//отменяем стандартное поведение браузера

// const link = document.querySelector('a');

// link.addEventListener('click', (event) => {

//     event.preventDefault();

// });

// навешиваем обработчик события на все кнопки через forEach

// btns.forEach(item => {

//     item.addEventListener('click', deleteElement);

// });

// обработчик события страбатывает только 1 разб с помощью once: true

btns.forEach(item => {

    item.addEventListener('click', deleteElement, {once: true});

});

**32. Навигация по DOM - элементам, data-атрибуты,  
преимущество for/of**

**37. ClassList и делегирование событий**

const btns = document.querySelectorAll('button');

//console.log(btns[0].classList.length); // узнаем количество классов у элемента

//console.log(btns[0].classList.item(0)); // узнаем с помощью метода "item" какой класс у элемента под индексом 0

btns[0].classList.add('red');   /// добавляет класс

btns[0].classList.add('red', '12345');   /// можно добавлять, удалять несколько классов

btns[0].classList.remove('blue');  // удаляет класс

btns[0].classList.toggle('blue'); //позволяет доглить, т.е. убирает класс, если есть и наоборот добавляет если нет

// пример гамбургера - убирает и показывает меню

// btns[0].addEventListener( 'click', () => {

// if (!btns[1].classList.contains('red')) {      // проверяем с помощью contains ----если ли класс

//     btns[1].classList.add('red');              // если нет, то добавляем класс

// } else {

//     btns[1].classList.remove('red');    // если есть, удаляем

// }

// });

//тоже самое с помощью toggle

// btns[0].addEventListener( 'click', () => {

//     btns[1].classList.toggle('red');

//     });

    //ДЕЛЕГИРОВАНИЕ СОБЫТИЙ

    const wrapper = document.querySelector('.btn-block');

    // wrapper.addEventListener('click', (event) => {   // оболочка кнопок

    //     if (event.target.tagName = "BUTTON") {    // делегируем какие-то действия на кнопки

    //         console.log('Hello');

    //     }

    // });

    wrapper.addEventListener('click', (event) => {   // оболочка кнопок

        if (event.target.classList.contains('blue')) {    // делегируем с помощью "contains" на кнопки у которых есть класс 'blue'

            console.log('Hello');

        }

    });

    //добавляем кнопку динамически

    const btn = document.createElement('button'); //создаем элемент на странице

    btn.classList.add('red', 'blue'); // добавляем классы

    wrapper.append(btn); // вставляем на страницы в конец

**38. Создаем табы в новом проекте**

const tabs = document.querySelectorAll('.tabheader\_\_item'),

        tabsContent = document.querySelectorAll('.tabcontent'),

        tabsParent = document.querySelector('.tabheader\_\_items'),

        price = document.querySelectorAll('.tabheader\_\_item\_\_price');

    function hideTabs() {

        tabsContent.forEach(item => {

            item.classList.add('hide');

            item.classList.remove('show', 'fade');

        });

        tabs.forEach(item => {

            item.classList.remove('tabheader\_\_item\_active');

        });

    }

        function showTabs(i = 0) {

            tabsContent[i].classList.add('show', 'fade');

            tabsContent[i].classList.remove('hide');

            tabs[i].classList.add('tabheader\_\_item\_active');

        }

    hideTabs();

    showTabs();

    const tabsStart = tabsParent.addEventListener('click', (e)=> {

        if (e.target && e.target.classList.contains('tabheader\_\_item')) {

            tabs.forEach((item, n) => {

                if (e.target == item) {

                    hideTabs();

                    showTabs(n);

                }

            });

        }

    });

**39. Скрипты и время их выполнения. setTimeout и setInterval**

// const timerId = setTimeout(function(){ // setTimeout - выполняется 1 раз

//     console.log('Hello');

// }, 2000);

// const timerId = setTimeout(function(text){

//     console.log(text);

// }, 2000, 'Hello');

// const timerId = setTimeout(logger, 2000); //передаем функцию logger

// function logger() {

//     console.log('text');

// }

const btn = document.querySelector('.btn');

let timerId,

    i = 0;

btn.addEventListener('click', () => {

    //const timerId = setTimeout(logger, 2000);

    timerId = setInterval(logger, 500);

});

function logger() {

    if ( i === 3) {

        clearInterval(timerId);

    }

         console.log('text');

         i++;

     }

    // рекрусивный вызов setTimeout

let id = setTimeout(function log() {

    console.log('Hello');

    id = setTimeout(log, 500);

}, 500)

**45. Функции-конструкторы**

function User(name, id) {

*this*.name = name;

*this*.id = id;

*this*.human = true;

*this*.hello = () => {

        console.log(`Привет ${*this*.name}`);

    };

}

User.prototype.exit = function () {

    console.log(`Пользователь ${*this*.name} ушел`);

};

const ivan = new User('Ivan', 28);

const petr = new User('Petr', 30);

ivan.exit();

console.log(ivan);

console.log(petr);

ivan.hello();

petr.hello();

**46. Контекст вызова. This**

use strict';

// 1) Обычная функция: This = windows, но если use strict = undefined

// function showThis() {

//     console.log(this);

// }

// showThis();

// function showThis(a, b) {

//     console.log(this);

//     function sum() {

//         console.log(this);

//         return this.a + this.b;

//     }

//     console.log(sum());

// }

// showThis(4, 10);

// 2)Контекст у методов объекта - сам объект

// const obj = {

//     a: 20,

//     b: 15,

//         sum: function() {

//             console.log(this);

//         }

// };

// obj.sum();

// 3) this в констуктарах и класах - это новый экземпляр объекта

// function User(name, id) {

//     this.name = name;

//     this.id = id;

//     this.human = true;

//     this.hello = () => {

//         console.log(`Привет ${this.name}`);

//     };

// }

// let ivan = new User('Ivan', 23);

// console.log(ivan);

// 4) ручная привязка this: call, apply и bind

// function sayName (surName) {

//     console.log(this);

//     console.log(this.name + ', ' + surName);

// }

// const user = {

//     name: 'John'

// };

// sayName.call(user, 'Smith');

// sayName.apply(user, ['Smith']);

// function count(num) {

//     return this \* num;

// }

// const double = count.bind(2);

// console.log(double(15));

// console.log(double(45));

const btn = document.querySelector('button');

    btn.addEventListener('click', function() {

        console.log(*this*);

    });

// ОЧЕНЬ ВАЖНО    стрелочная функция не имеет контекста THIS, поэтому она обращается к функции выше, а функция обращается к NUM, если бы вместо стрелочной функции была обычная было бы undefined

const obj = {

    num: 10,

    sayNumber: function () {

        const say = () => {

            console.log(*this*.num);

        };

        say();

    }

};

obj.sayNumber();

**47. Классы (ES6)**

'use strict';

class Rectangle {

    constructor(height, width) {

*this*.height = height;

*this*.width = width;

    }

    calcArea () {

        return *this*.height \* *this*.width;

    }

}

// const square = new Rectangle(10, 10);

// const long = new Rectangle(10, 8);

// console.log(square.calcArea());

// console.log(long.calcArea());

class ColoredRectangleWitText extends Rectangle { //наследуется из Rectangle

    constructor(height, width, text, bgColor) {

*super*(height, width);

*this*.text = text;

*this*.bgColor = bgColor;

    }

    showMyProps() {

        console.log(`Текст: ${*this*.text}, цвет: ${*this*.bgColor}`);

    }

}

const div = new ColoredRectangleWitText (25, 10, 'Hello world', 'red');

div.showMyProps();

console.log(div.calcArea);

**48. Практика**

**49. Rest оператор и параметры по умолчанию (ES6)**

// rest оператор

const log = function (a, b, ...rest) { // rest оператор обозначается ... после 3 точек можно указывать любое значение

    console.log(a, b, rest);

};

log('basic', 'rest', 'operator', 'user'); // rest оператор собрал другие значения в массив пример: basic rest [ 'operator', 'user' ]

// параметры по умолчанию

// function calcOrDouble(num, basis) {

//     basis = basis || 2;           // параметр по умолчанию до ES6

//     console.log(num \* basis);

// }

// calcOrDouble(3);

function calcOrDouble(num, basis = 2 ) { // параметр по умолчанию ES6

    console.log(num \* basis);

}

calcOrDouble(3);

**+ ПРАКТИКА**

**51. JSON формат передачи данных, глубокое клонирование  
объектов**

// const persone = {

//     name: 'Alex',

//     tel: '89004331212'

// };

// console.log(JSON.parse(JSON.stringify(persone)));// stringify отправляет данные на сервер (бэкенд)

                                                 // parse получает данные с сервера ( на фронтенд)

// глубокое копирование ---клонироание с помощью JSON.parse(JSON.stringify())

const persone = {

    name: 'Alex',

    tel: '89004331212',

    parents: {

        mom: 'Olga',

        dad: 'Mike'

    }

};

const clone = JSON.parse(JSON.stringify(persone));

clone.parents.mom = 'Anna';

clone.name = 'Max';

console.log(persone);

console.log(clone);

**52. AJAX и общение с сервером** **ПРАКТИКА**

**CALC пример**

const inputRub = document.querySelector('#rub'),

    inputUsd = document.querySelector('#usd');

    inputRub.addEventListener('input', () => {

        const inquiry = new XMLHttpRequest(); // первый способ для работы с асинхронным кодом

        //методы XMLHttpRequest()

        // первый метод open(metod, url, async, login, password)

        inquiry.open('GET', 'js/current.json');

        inquiry.setRequestHeader('Content-type', 'application/json', 'charset=utf-8'); // заголовок HTTP

        inquiry.send(); // отправка запроса (метод send())

        // свойства

        // 1. status (200-ок)

        // 2. statusText

        // 3. response (ответ)

        // 4. readyState (текущее состояние запроса 4-ок)

        // События

        // 1. readystatechange

        // inquiry.addEventListener('readystatechange', () => { // это событие отслеживает статус готовности запроса

        //     if (inquiry.readyState === 4 && inquiry.status === 200) {

        //         const data = JSON.parse(inquiry.response); // получаем ответ(response) от сервера js/current.js

        //         inputUsd.value = (+inputRub.value / data.current.usd).toFixed(2);

        //       } else {

        //         inputUsd.value = "Что-то пошло не так";

        //       }

        // });

        // 2. load

        inquiry.addEventListener('load', () => { // это событие отслеживает статус готовности запроса

          if (inquiry.status === 200) {

              const data = JSON.parse(inquiry.response); // получаем ответ(response) от сервера js/current.js

              inputUsd.value = (+inputRub.value / data.current.usd).toFixed(2);

            } else {

              inputUsd.value = "Что-то пошло не так";

            }

      });

    });

**53. Реализация скрипта отправки данных на сервер**

**54. Красивое оповещение пользователя**

//Forms

    const forms = document.querySelectorAll('form'); // получаем все формы по тэгу form

    const message = {

        loading: 'img/form/spinner.svg',

        success: 'Спасибо! Скоро мы с Вами свяжемся',

        failure: 'Ошибка'

    };

    forms.forEach(item => {

        postData(item);

    });

    function postData(form1) {

        form1.addEventListener('submit', (e) => {

            e.preventDefault();

            const statusMessage = document.createElement('img');

            statusMessage.src = message.loading;

            statusMessage.style.cssText = `

                display: block;

                margin-top: 10px;

                margin: 0 auto;

            `;

            //form1.append(statusMessage); убираем так как спинер встает

            // не посередине формы а в конец формы

            form1.insertAdjacentElement('afterend', statusMessage); // insertAdjacentElement вставляет div спинер после формы (посередине)

            const r = new XMLHttpRequest();

            r.open('POST', 'server.php');

            // 1 способ отправки данных formData

            // const formData = new FormData(form1);

            // r.send(formData);

            // r.addEventListener('load', () => {

            //     if (r.status === 200) {

            //         console.log(r.response);

            //         statusMessage.textContent = message.success;

            //         form1.reset(); // очищаем форму после отправки

            //         setTimeout( () => {

            //             statusMessage.remove();

            //         }, 2000);       // удаляем сообщение statusMessage

            //      } else {

            //         statusMessage.textContent = message.failure;

            //     }

            // });

            // 2 способ отправки данных formData

            r.setRequestHeader('Content-type', 'application/json');

            const formData = new FormData(form1);

            const object = {};    // прием - помещаем данные из formData в обычный объект

            formData.forEach( (value, key) => {

                object[key] = value;

            });

            const json = JSON.stringify(object);

            r.send(json);

            r.addEventListener('load', () => {

                if (r.status === 200) {

                    console.log(r.response);

                    showThanksModal(message.success);

                    form1.reset(); // очищаем форму после отправки

                    setTimeout( () => {

                        statusMessage.remove();

                    }, 2000);       // удаляем сообщение statusMessage

                 } else {

                    showThanksModal(message.failure);

                }

            });

          });

    }

    function showThanksModal(message) {

        const modalDialog = document.querySelector('.modal\_\_dialog');

        modalDialog.classList.add('hide');

        openModal();

        const thanksModal = document.createElement('div');

        thanksModal.classList.add('modal\_\_dialog');

        thanksModal.innerHTML = `

            <div class="modal\_\_content">

            <div class="modal\_\_close" data-close>×</div>

            <div class="modal\_\_title">${message}</div>

        </div>

        `;

        const modalAppend = document.querySelector('.modal');

        modalAppend.append(thanksModal);

        setTimeout(() => {

            thanksModal.remove();

            modalDialog.classList.add('show');

            modalDialog.classList.remove('hide');

            closeModal();

        }, 4000);

    }

});

**55. Promise (ES6)**

console.log('Запрос данных...');

const reg = new Promise((resolve, reject) =>{  //Promise - обещание, принимает 2 аргумента

                                                    // resolve - обещание выполнилось

                                                    //reject - обещание не выполнилось

    setTimeout( () => {

    console.log('Получение данных...');

    const product = {

        name: 'TV',

        price: 500

    };

    resolve(product);

    //reject(); // выполняется если произошла ошибка методом catch - ставится в конце

}, 2000);

});

    reg.then((data) => { //метод then - отрабатывает положительный результат resolve

        return new Promise((resolve, reject) => {

            setTimeout( () =>{

                data.status = 'order';

                resolve(data);

            }, 1000);

        });

    }).then(data => {

        data.percent = "50%";

        return data;

    }).then(data2 => {

        console.log(data2);

    }).catch(() =>{

        console.error('Произошла ошибка');

    }).finally(() =>{     // метод finally -- ставится в конце, позволяет выполнить действия

                          // при любом исходе промиса.

        console.log('finally');

    });

// метод Promise.all и метод Promise.race

const test = (time) => {

    return new Promise ((resolve, reject) => {

       setTimeout( () =>{

        resolve();

       },time);

    });

};

// test(1000).then(() => {

//     console.log('1000');

// });

// test(2000).then(() => {

//     console.log('2000');

// });

Promise.all([test(1000), test(2000)]).then(() =>{  // all - все промисы выполнились, т.е. ждем загрузки всех

                                                   // промисо и только после этого выполнить какие-то действия

    console.log('All');

});

Promise.race([test(1000), test(2000)]).then(() =>{  // race - гонка, ждем когда выполниться самый первый промис

console.log('Race');

});