|  |  |
| --- | --- |
| <div class="sell\_\_text">     <ul class="sell\_\_list">       <li> <span></span>с августа, сразу после выкачки, - мед разнотравье </li>       <li>  <span></span>с декабря - инкубационное яйцо </li>     </ul>  </div>  > никак не ставился вертикально по центру, добавила в css высоту строки и шрифт. | .sell\_\_list{    display: flex;    flex-direction: column;    align-items: start;}  .sell\_\_list li{    text-align: center;}  .sell\_\_list li span {    vertical-align: middle;    display: inline-block;    width: 20px;    height: 20px;    color: #eddd5e;    background-color: #5b8c51;    border-radius: 50%;    line-height: 18px;    font-size: 12px;}  .sell\_\_list li span::before {    content: ">";} |

Открыть рабочий проект в новом компе

1. **Установить плагины VSCode**

Список необходимых плагинов:

1. All Autocomplete

2. Auto Close Tag

3. Auto Complete Tag

4. Auto Rename Tag

5. Beautify

6. Code Runner

7. GitLens — Git supercharged

8. Import Cost

9. JavaScript (ES6) code snippets

10. jshint

11. Live Server

12. Multiple clipboards for VSCode

13. Path Autocomplete

14. Reactjs code snippets

15. Sass

16. Theme - Oceanic Next

17. vscode-icons

18. ESLint

1. Установить Git
2. Установить Sass, Live Sass

В параметрах (шестеренка внизу слева или ctrl+) открыть файл Json (вверху справа лист со стрелкой вокруг) добавить

"liveSassCompile.settings.formats": [

{

"format": "expanded",

"extensionName": ".css",

"savePath": "/css"

}

],

В папке css/ style.css.map прописать путь

   "sources": [

        "../scss/style.scss"

    ],

    "file": "style.css"

Проверить наличие CSS класса

[classList.contains](https://developer.mozilla.org/ru/docs/Web/API/Element/classList) вместо *className*

**contains** ( String )  
Проверяет, есть ли данный класс у элемента (вернет true или false)

использовать можно так:

if(this.classList.contains('active'))

где *this* должен быть html-элементом.

через npm install @rollup/plugin-babel --save-dev устанавливаем пакет в зависимости,  
через import import { babel } from '@rollup/plugin-babel' подключаем к проекту.  
и в конфиге в свойстве plugins указываем настройки [babel({ babelHelpers: 'bundled' })].

npm init -y это инициализация проекта точнее создание файла package.json, а npm i или npm install это установка всех зависимостей которые мы указали в файле package.json

для того чтоб появилась папка dist нужно вызвать команду npm run prod , и соберётся готовое приложение.

Если у вас при установке @rollup/plugin-babel и других пакетов возникают ошибки (нераспознанный токен в исходном тексте), то заключите название пакета в кавычки:

npm install "@rollup/plugin-babel" --save-dev

Также обратите внимание!

Для того, чтобы была конвертация JS кода в старую версию, также на 04:25 необходимо добавить пресет [@babel/preset-env](https://babeljs.io/docs/en/babel-preset-env):

npm i -D @babel/preset-env

И в rollup.config.js в plugins необходимо его подключить:

plugins: [

babel({

babelHelpers: "bundled",

presets: ["@babel/env"]

})

];

Если вы используете current версию Node.js и у вас появляются ошибки при использовании webpack-dev-server или других пакетов, то попробуйте переустановить Node.js на версию LTS, удалить папку node\_modules и еще раз установить зависимости (команда npm install)

## [Округление](https://learn.javascript.ru/number" \l "okruglenie)

Одна из часто используемых операций при работе с числами – это округление.

В JavaScript есть несколько встроенных функций для работы с округлением:

**Math.floor**

Округление в меньшую сторону: 3.1 становится 3, а -1.1 — -2.

**Math.ceil**

Округление в большую сторону: 3.1 становится 4, а -1.1 — -1.

**Math.round**

Округление до ближайшего целого: 3.1 становится 3, 3.6 — 4, а -1.1 — -1.

**Math.trunc (не поддерживается в Internet Explorer)**

Производит удаление дробной части без округления: 3.1 становится 3, а -1.1 — -1.

Ниже представлена таблица с различиями между функциями округления:

|  | **Math.floor** | **Math.ceil** | **Math.round** | **Math.trunc** |
| --- | --- | --- | --- | --- |
| 3.1 | 3 | 4 | 3 | 3 |
| 3.6 | 3 | 4 | 4 | 3 |
| -1.1 | -2 | -1 | -1 | -1 |
| -1.6 | -2 | -1 | -2 | -1 |

Функция **isNaN()** определяет, является ли литерал или переменная нечисловым значением ([NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN)) или нет.

isNaN(NaN); // true

isNaN(undefined); // true

isNaN({}); // true

isNaN(true); // false

isNaN(null); // false

isNaN(37); // false

**// strings**

isNaN("37"); // false: "37" преобразуется в число 37 которое не NaN

isNaN("37.37"); // false: "37.37" преобразуется в число 37.37 которое не NaN

isNaN(""); // false: пустая строка преобразуется в 0 которое не NaN

isNaN(" "); // false: строка с пробелом преобразуется в 0 которое не NaN

isNaN("37,5"); // true

**// Даты**

isNaN(new Date()); // false

isNaN(new Date().toString()); // true

**// Пример почему использование isNaN не всегда уместно**

isNaN("blabla") // true: "blabla" преобразовано в число.

// При парсинге преобразуется в число при неудаче возвращает NaN

Глобальное свойство NaN является значением, представляющим не-число (Not-A-Number).

Обратите внимание на разницу между функцией isNaN() и методом Number.isNaN(): первая вернёт true, если значение в настоящий момент является NaN, или если оно станет NaN после того, как преобразуется в число, в то время как последний вернёт true, только если текущим значением является NaN:

isNaN('hello world'); // true

Number.isNaN('hello world'); // false

Метод **Number.isNaN(***value***)** определяет, является ли переданное значение [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN)

*value* - значение, проверяемое на [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN).

В отличие от глобальной функции [isNaN()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/isNaN), Number.isNaN() не имеет проблемы принудительного преобразования параметра в число. Это значит, что в него безопасно передавать значения, которые обычно превращаются в [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN), но на самом деле [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN) не являются. Также это значит, что метод возвращает true только для числовых значений, имеющих значение [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN).

Number.isNaN(***x***) будет надёжным методом для проверки, содержит ли  ***x***значение NaN или нет. Даже с Number.isNaN, однако, результатом NaN остаётся точное числовое значение, а не просто "not a number".

Number.isNaN(NaN); // true

Number.isNaN(Number.NaN); // true

Number.isNaN(0 / 0) // true

**// При использовании глобальной функции isNaN() это всё будет true**

Number.isNaN('NaN'); // false

Number.isNaN(undefined); // false

Number.isNaN({}); // false

Number.isNaN('blabla'); // false

**// А это всё в любом случае будет false**

Number.isNaN(true); // false

Number.isNaN(null); // false

Number.isNaN(37); // false

Number.isNaN('37'); // false

Number.isNaN('37.37'); // false

Number.isNaN(''); // false

Number.isNaN(' '); // false