***JavaScript*** – язык программирования, используется для создания логики сайта и придания ему интерактивности. Для работы рекомендуется использовать приложение *VSCode* и *node.js*. В *VSCode* устанавливаем расширение *code runner* для быстрого запуска и настраиваем файл *User Settings (JSON)*:

*{ "code-runner.showExecutionMessage": false,*

*"code-runner.clearPreviousOutput": true,*

*"editor.minimap.enabled": false,*

*"files.autoSave": "afterDelay" }*

Закомментировать код можно с помощью ***//*** или ***/\* код \*/***

***<script> </script>*** - теги элемента-скрипта. Рекомендовано помещать перед *</body>*. Атрибут ***src*** указывает путь к файлу со скриптом.

*<script src="script.js"></script>*

Для передачи значения переменной используется синтаксис ***${имя\_переменной}***. Если значение переменной требуется заменить в строке, то кавычки заменяются на апостроф ***`***.

*<img src={`https://randomuser.me/api/portraits/${props.gender}/78.jpg`}/>*

Перейдем к этому файлу *JS*.

Для вывода на страницу используется команда ***document.write()***.

*document.write('В чём сила?');*

Для вывода в браузерную консоль используется команда ***console.log()***.

*console.log('Сила — в правде.');*

*console.log((2 + 2) / 2);*

*JS* позволяет работать с числами (целыми и дробными) и строками. При выводе строки всегда заключаются в кавычки (одиночные или двойные не имеет значения). При сложении двух строк произойдет конкатенация. При сложении числа и строки, число будет интерпретироваться как строка.

Точка с запятой в конце строки не является обязательными.

В старых версиях *JS* для создания переменной использовалось ключевое слово ***var***, однако его престали использовать из-за эффекта *hoisting* (интерпритатор просматривает код и создает все переменные перед его выполнением). Например, переменная-счетчик цикла *i* была доступна и хранила значение после его завершения.

Для создания переменной используется ключевое слово ***let***. Оно не имеет *hoisting* эффект. Переменные не имеют типов. По умолчанию значение переменной не определено – ***undefined***.

*let text = “new text”; // создавать переменную и задавать ей значение можно сразу в одну строчку*

*let pages; // или в несколько*

*pages = 210;*

Ключевое слово ***const*** фиксирует тип и значение переменной и не позволяет их изменить. При использовании на типе объект, разрешается добавлять/удалять/изменять его вложенные свойства.

*const a = 10;*

В приоритете рекомендуется использовать *const*, а не *let*.

**Массив** – структура данных, в которой однородные элементы перечисляются через запятую и имеют порядковый номер (**индекс**). Нумерация индекса идет с нуля.

*const aliExpress = ['Лазерная указка Xioamo', 'Форма для льда «Титаник»'];*

*aliExpress[0]; // 'Лазерная указка Xioamo'*

*AliExpress.length; // получить длину массива*

*aliExpress.push(‘Что-то еще’); // добавить элемент в массив*

*aliExpress.splice(2, 1); // удалить один элемент с индексом 2*

*const numbers = [1, 2, 3, 4, 5];*

*const numberTimesTwo = numbers.map(number => number \* 2); // удваиваем числа в массиве*

*console.log(numberTimesTwo);*

*// массовое извлечение переменных из массива в порядке возрастания индекса*

*// в переменную another запишутся все оставльные элементы в виде массива*

*const [num0, num1, num2, …another] = numbers;*

Для получения случайного числа используется метод ***Math.random()***, возвращающий дробное число от [0 до 1) (не вкл):

*let randomNumber = Math.random();*

Чтобы округлить дробное число до целого, используется метод ***Math.floor()***:

*let randomNumberInt = Math.floor(Math.random());*

*JS* позволяет работать с **циклами** *for*, *while*, *do-while*.

*JS* позволяет работать с **условными операторами** *if-else* и *switch*.

Инкремент и декремент поддерживаются.

*let array = ['животные', 'растения', 'грибы', 'микроорганизмы', 'вирусы'];*

*for (let i = 0; i <= 4; i = i + 1) {*

*console.log(array[i]);*

*}*

При сравнении переменных через ***==*** JS принудительно приводит обе переменные к единому типу для сравнения. Чтобы избежать этого, нужно использовать ***===***.

Для хранения разнородных данных используются **объекты** (аналог *json*). Они создаются фигурными скобками. Объекты имеют свойства, которые состоят из ***ключ: значение***. Обратиться к свойству объекта можно через точку.

*let user = {*

*name: 'Мария',*

*dotaLevel: 21,*

*dogName: 'Железногорск',*

*toString: function() { // функция внутри объекта*

*return `Name: ${this.name}`*

*}*

*};*

*console.log(user.name); // Мария*

*console.log(Object.keys(user)); // вывести все ключи объекта*

*console.log(Object.values(user)); // вывести все значения объекта*

*delete user.dogName; // удалить свойство*

*const { name, dotaLevel } = user; // массовое извлечение свойств в отдельные константы*

*const person = { // создание объекта на основании других объектов*

*…user, // копируем все свойства user*

*address: {…address} // вложенный объект со свойствами adress*

*};*

**Функция** – это блок кода, который можно переиспользовать много раз. Для её создания используют ключевое слово ***function***. Функция может принимать параметры. Для возврата из функции используется ключевое слово ***return***.

*function sayHello(name) {*

*return 'Привет, ' + name;*

*}*

*let greeting = sayHello(‘Андрей’); // вызов функции*

*alert(greeting); // вывод отдельное окно с сообщением поверх страницы*

*const sayHello = function(name) { // другие способы задания функции*

*return 'Привет, ' + name;*

*}*

*const sayHello = name => ( // другие способы задания функции*

*'Привет, ' + name;*

*)*

*const sayHello = name => 'Привет, ' + name; // другие способы задания функции*

*const sayHello = (name = “default”) => 'Привет, ' + name; // задаем параметр функции по умолчанию*

**Обратная функция, *callback*** – это блок кода, позволяющий использовать функцию без её объявления.

*const greetUser = (username, callback) => {*

*if (callback) {*

*console.log(callback(username);*

*} else {*

*console.log(username); // вызов, если функция не задана*

*}*

*}*

*greetuser(“Jamila”, (u) => {*

*return “Hello “ + u;*

*});*

*JS* поддерживает импорт/экспорт модулей. Для включения этих функций нужно инициировать проект - в папке проекта запустить в терминале команду:

*npm init -y*

После этого в корне проекта создастся файл *package.json* с его настройками. Добавим строку:

*“type”: “module” // по умолчанию используется commonjs*

После этого заработают ключевые слова import/export:

*const BRAND = "test";*

*const add = (a, b) => a + b;*

*const substract = (a, b) => a - b;*

*export default BRAND; // экспортируемая функция по умолчанию*

*export { // именованные экспорты*

*add,*

*substract,*

*BRAND as brand // переименовываем экспорт*

*}*

*import*

*brand, // импортируемая функция по умолчанию (имя любое)*

*{ add, substract, BRAND as b }, // именованные импорты с переименованием*

*\* As Lib // ИЛИ импортировать как библиотеку*

*from "./lib.js";*

*console.log(brand);*

*console.log(add(10, 10));*

*console.log(Lib.substract(10, 10));*

Чтобы работать с элементами страницы, в *JS* есть специальный ящик ***document***. В нем находится вся информация о веб-странице: *URL*-адрес, таблицы стилей, кодировка, гиперссылки, все тексты. Через него можно получить доступ к управлению любым элементом на странице.

Инструмент ***querySelector*** принимает на вход селектор класса элемента, к которому мы хотим получить доступ, и возвращает это значение для записи в переменную:

*let photoElement = document.querySelector('.photo');*

Инструмент ***textContent*** позволяет заменить текущий текст элемента:

*phrase.textContent = getRandomElement(phrases);*

Инструмент ***advice.style.fontSize*** позволяет изменить размер шрифта элемента *advice.*

Инструмент ***image.src*** позволяет изменить ссылку на картинку элемента *image.*

Инструмент ***addEventListener*** позволяет добавить каждому элементу реакцию на событие.

*// находим элемент .button и кладём в переменную*

*let button = document.querySelector('.button');*

*// обращаемся к переменной, добавляем элементу слушатель клика*

*button.addEventListener('click', function () {*

*// присваиваем новую фразу переменной с элементом phrase*

*phrase.textContent = getRandomElement(phrases);*

*// меняем размер шрифта в зависимости от длины текста*

*if (randomElement.length > 40) {*

*advice.style.fontSize = '33px';*

*} else {*

*advice.style.fontSize = '42px';*

*}*

*});*

Для плавности анимации можно подключить стороннюю библиотеку:

*<script src="https://code.s3.yandex.net/web-code/smoothly.js"></script>*

После подключения библиотеки в коде станет доступна функция ***smoothly***. Она принимает три аргумента:

* элемент, к которому собираетесь применить эффект
* свойство элемента, которое должно плавно измениться
* новое значение свойства

*let element = document.querySelector('.element');*

*smoothly(element, 'textContent', 'Новое содержимое');*

*// в нашем примере это будет выглядеть так*

*smoothly(phrase, 'textContent', randomElement.text);*

*smoothly(image, 'src', randomElement.image);*

Интерфейс ***Promise*** представляет собой объект, который содержит свое состояние (*pending*, *fulfilled* или *rejected*). При изменении состояния вызывается *callback*-функция либо *onFulfilled* (успех), либо *onRejected* (ошибка). Функции внутри *promise* запускаются **асинхронно** между собой.

*const promise = new Promise((resolve, reject) => {*

*setTimeout(() => { // переведёт в состояние fulfilled с результатом " success"*

*resolve("success");*

*}, 1000);*

*});*

*promise.then(result => { // запустится при вызове resolve*

*alert("Fulfilled: " + result); // Fulfilled: success*

*},*

*error => { // запустится при вызове reject*

*alert("Rejected: " + error);*

*});*

Такой код **БУДЕТ блокировать** выполнение потока до окончания вычислений:

*const promise = new Promise((resolve, reject) => {*

*let i = 0;*

*while (i < 10\_000\_000\_000) { i++ }*

*resolve("data back from the server")*

*});*

Такой код **НЕ БУДЕТ блокировать** выполнение потока до окончания вычислений:

*const promise = new Promise((resolve, reject) => {*

*return Promise.resolve().then(() => {*

*let i = 0;*

*while (i < 10\_000\_000\_000) { i++ }*

*resolve("data back from the server")*

*})*

*});*

При наличии множества функций, которые мы хотели бы сделать асинхронными, *promise* получился бы слишком громоздким. Альтернативой является использования функции ***async()***, которая всегда возвращает *promise*.

*const customers = new Promise((resolve, reject) => { // первый promise*

*return Promise.resolve().then(() => {*

*let i = 0;*

*while (i < 2\_000\_000\_000) { i++ }*

*resolve([*

*{name: 'Jamila', id: 1},*

*{name: 'Bob', id: 1}*

*])*

*})*

*});*

*const adresses = new Promise((resolve, reject) => { // второй promise*

*return Promise.resolve().then(() => {*

*let i = 0;*

*while (i < 2\_000\_000\_000) { i++ }*

*resolve([*

*{customerId: 1, adress: "Jamila"},*

*{customerId: 2, adress: "USA"}*

*])*

*})*

*});*

*const fetchData = async() => { // запускаем асинхронно без блокировки потока*

*try {*

*const c = await customers; // ждем выполнения первого promise*

*const a = await adresses; // ждем выполнения второго promise*

*console.log(c);*

*console.log(a);*

*} catch (error) {*

*co****n****sole.log(error);*

*}*

*}*

*fetchData();*

В ситуации, когда нам нужно получить все или ничего, можно использовать ***promise.all()***:

*const fetchData = async() => {*

*Promise.all([customers, adresses]).then(values => {*

*console.log(values)*

*}).catch (error => {*

*console.log(error);*

*})*

*}*

Клиент ***AXIOS*** позволяет работать с *promise* на *http* уровне. Установить его можно командой:

*npm install axios*

Рассмотрим пример использования *axios*:

*import axios from "axios";*

*const fetchJoke = async(url) => {*

*try {*

*const res = await axios.get(url);*

*console.log(res.data);*

*} catch (error) {*

*console.log(err);*

*}*

*}*

*fetchJoke("https://api.chucknorris.io/jokes/random");*