|  |
| --- |
| Блок 1. Основы JavaScript  Занятие 2. JSON, работа с объектами и массивами.1) Сконструировать JSON с сотрудниками, их телефонами и адресами Информация о сотруднике должна включать id, name (имя) и surname (фамилию) сотрудника.  Поместить в файл **employees-json.js**:  **const *DATA*** = {   **employees**: [  {  **id**: 1,  **name**: **"Пафнутий"**,  **surname**: **"Пафнутьев"**,  **department**: **"IT"** }  ] }  Добавить информацию о нескольких сотрудниках. 2) Подключить скрипт и JSON к файлу HTML Создать файл index.html, используя стандартный шаблон WebStorm.  В конце прописать:  <script src="employees-json.js"></script>  <script src="service.js"></script>  Создать файл service.js, в который будут добавляться функции. 3) Реализовать функцию для поиска сотрудника по имени и фамилии (возвращать массив). Разработать функцию ***findByName*(name, surname)**  Функция находит сотрудника по его имени. В случае, если имя или фамилия пустые, они игнорируются.  Например, findByName("","") находит всех сотрудников. findByName("Иван") находит всех Иванов. findByName(null,"Иванов") находит всех Ивановых.  Использовать цикл for (лучше for ... of) для перебора сотрудников.  **function** *findByName*(name, surname) {  **let** res = [];  **for** (**var** e **of *DATA***.**employees**) {  **if** ((!name || e.**name**===name) &&  (!surname || e.**surname**===surname)) {  res.push(e);  }  }  **return** res; } 4) Реализовать функцию для добавления сотрудника по имени и фамилии. Функция добавляет сотрудника по имени. id присваивается автоматически, как самый большой id среди сотрудников + 1. В случае, если имя или фамилия не заданы, функция выбрасывает исключение с сообщением об ошибке.  ***addEmployee(name, surname)***  **function** *addEmployee*(name, surname) {  **if** (!name || name.**length**==0 || !surname || surname.**length**==0) {  **throw new *Error***(**"name and surname should be not empty"**);  }  **let** max = 0;  **for** (**let** e **of *DATA***.**employees**) {  **if** (e.**id**>max) max = e.**id**;  }  **let** id = max+1;  ***DATA***.**employees**.push({id,name,surname});  **return** id; } 5) Разработать функцию для удаления сотрудника по id. Использовать for...of для поиска сотрудника.  Использовать метод массива splice(index, amount) для удаления.  **function** *removeEmployee*(id) {  **let** index = 0;  **for** (**let** e **of *DATA***.**employees**) {  **if** (e.**id**===id) **break**;  index++;  }  ***DATA***.**employees**.splice(index, 1); } 6) Разработать функцию для вывода в консоль всей информации по сотруднику. Функция showEmployee(employee) должна выводить в консоль всю информацию о сотруднике. Для этого функция должна использовать метод Object.keys для получения всех полей объекта employee. Далее она должна выводить информацию в формате ключ=значение в консоль, используя метод console.log().  **function** *showEmployee*(employee) {  **const** keys = ***Object***.keys(employee);  ***console***.log(**"Информация о сотруднике "**+employee[**"name"**]+**":"**);  **for** (**let** key **of** keys) {  ***console***.log(key+ **" = "**+employee[key]);  } } 7) Реализовать функцию, выводящую список всех сотрудников. Функция showEmployees() должна брать список всех сотрудников из JSON и выводить информацию по каждому, используя функцию showEmployee(employee).  Для перебора можно использовать for...of.  Альтернативно можно использовать forEach.  **function** *showEmployees*() {  *// альтернативный вариант:*  *// DATA.employees.forEach(showEmployee);* **for** (**let** e **of *DATA***.**employees**) {  *showEmployee*(e);  } } 8) Проверять работу функций в консоли Открыть консоль в браузере и запустить разные функции для проверки их работоспособности. |
|  |

## Занятие 3. Работа со строками и датами.

### 1) Написать функцию, которая будет находить сотрудника по id

Функция findById(id) должна перебирать всех сотрудников и находить сотрудника с совпадающим id.

**function** *findById*(id) {  
 **for** (**var** e **of *DATA***.**employees**) {  
 **if** (id==e.**id**) {  
 **return** e;  
 }  
 }  
}

### 2) Оптимизировать поиск по id: использовать объект для кэширования.

Создать объект employeeMap={} и сохранять там уже найденных сотрудников. Использовать id в качестве ключа и ссылку на сотрудника в качестве значения.

**const *employeeMap*** = {};  
  
**function** *findById*(id) {  
 **if** (***employeeMap***[id]) {  
 **return *employeeMap***[id];  
 }  
 **for** (**var** e **of *DATA***.**employees**) {  
 **if** (id==e.**id**) {  
 ***employeeMap***[id] = e;  
 **return** e;  
 }  
 }  
}

### 3) Реализовать метод для добавления телефона сотрудника.

*addPhone*(id, phone)

Список телефонов должен храниться внутри JSON-объекта сотрудник.

В качестве поля используется свойство phones типа массив.

Если такое поле у сотрудника отсутствует, оно должно быть создано.

**function** *addPhone*(id, phone) {  
 **const** employee = *findById*(id);  
 **const** phones = employee.**phones**;  
 **if** (!phones) {  
 employee.**phones** = [];  
 }  
 employee.**phones**.push(phone);  
}

### 4) Реализовать метод, устанавливающий дату рождения сотрудника

*setDateOfBirth*(id, date)

**function** *setDateOfBirth*(id, date) {  
 **const** employee = *findById*(id);  
 employee.**dateOfBirth** = date;  
}

### 5) Реализовать метод, возвращающий возраст сотрудника.

Функция getAge(id) принимает id сотрудника в качестве параметра и возвращает возраст сотрудника. Стоит отметить здесь, что в подобных случаях не стоит изобретать велосипед, и лучше нагуглить подходящее решение.

**function** *getAge*(id) {  
 **const** employee = *findById*(id);  
 **let** ageDiff = ***Date***.now() - employee.**dateOfBirth**.getTime();  
 **let** ageDate = **new *Date***(ageDiff); *// miliseconds from epoch* **return *Math***.abs(ageDate.getUTCFullYear() - 1970);  
}

### 6) Написать функцию, форматирующую дату и возвращающую ее в формате дд.мм.гггг.

*formatDate*(date)

Данная функция должна возвращать строку с датой. Для этого можно использовать методы класса Date: getDate(), getMonth(), getYear(). Для даты и месяца надо добавлять на первую позицию 0, если дата или месяц меньше 10.

**function** *formatDate*(date) {  
 **let** day = date.getDate();  
 **if** (day<10) day = **'0'**+day;  
 **let** month = date.getMonth()+1;  
 **if** (month<10) month = **'0'**+month;  
 **let** year = date.getFullYear();  
  
 **return** day + **'.'** + month + **'.'** + year;  
}

### 7) Написать функцию, возвращающую подробную информацию о сотруднике.

Функция *getEmployeeInfo*(id) должна возвращать строку, содержащую подробную информацию о сотруднике: имя, фамилию, дату рождения в отформатированном виде, возраст, список телефонов сотрудника.

**function** *getEmployeeInfo*(id) {  
 **const** e = *findById*(id);  
  
 **const** phones = e.**phones**?  
 **`Список телефонов:** ${e.**phones**}**`**:**''**;  
 **const** age = e.dateOfBirth?  
 **`Возраст:** ${*getAge*(e.**id**)}**`**:**''**;  
 **return `   
 Имя:** ${e.**name**} **Фамилия:** ${e.**surname**} **Дата рождения:** ${*formatDate*(e.dateOfBirth)}  
 ${phones}   
 ${age}  
 **`**;  
}

### 8) Написать тестовую функцию, которая добавляет сотрудника, добавляет телефоны, устанавливает дату рождения, и выводит подробную информацию о сотруднике в консоль.

Функция testEmployee() предназначена для тестирования функций с консоли.

**function** *testEmployee*() {  
 *addPhone*(133, **"555-55-55"**);  
 *addPhone*(133, **"666-66-66"**);  
 *setDateOfBirth*(133, **new *Date***(2000,1,1))  
 **const** info = *getEmployeeInfo*(133);  
 ***console***.log(info);  
}

### 9) Написать функцию, которая выводит информацию о сотруднике в формате JSON строки

Функция должна использовать метод JSON.stringify().

**function** *getEmployeeJSON*(id) {  
 **const** e = *findById*(id);  
 **return *JSON***.stringify(e);  
}