**Лабораторная работа №11**

**Тема работы:** Разработка программ с применением стрелочных функций

**Цель работы:** закрепить навыки по работе со стрелочными функциями.

**Теоретические сведения**

**Команда let вместо var**

Кроме команды **var** для объявления переменных появилась еще и команда **let**. Эта команда более строгая и нужна для того, чтобы ограничить область видимости переменных фигурными скобками.

Давайте посмотрим разницу между var и let. Посмотрите, как работает **var**: он не создает область видимости внутри **if** - если объявить переменную снаружи **if**, а потом переопределить ее внутри - снаружи эта переменная тоже поменяется:

var test **=** 5;

if (true) {

var test **=** 10;

alert(test); //введет 10

}

alert(test); //введет 10 - значение изменилось

А теперь посмотрим, как работает **let** - он создает разные области видимости снаружи и внутри **if**. И, если объявить переменную снаружи **if**, а затем попытаться затереть ее внутри - она не изменится:

let test **=** 5;

if (true) {

let test **=** 10;

alert(test); //введет 10

}

alert(test); //введет 5 - значение не изменилось

Команда **let** создает разные области видимости не только в **if**, но и в циклах и вообще в любых фигурных скобках.

**Константы**

В ES6 также появились константы. **Константы** - это такие переменные, значения которых нельзя изменить. Зачем нужны константы: в константу можно задать некоторое значение в начале работы скрипта - и можно будет гарантировать, что это значение в дальнейшем случайно не поменяется.

Константа объявляется командой **const** вместо команды **var**:

const test **=** 5;

Если попытаться сменить значение константы - мы получим ошибку:

const test **=** 5;

test **=** 10; //ошибка

**Константы-объекты**

Объекты также можно делать константами. В этом случае значения их свойств разрешено менять, так же, как и добавлять новые свойства:

const user **=** {

name: 'Иван'

};

user.name **=** 'Коля'; //можно менять

user.surname **=** 'Иванов'; //можно

А вот затереть объект и записать туда что-нибудь другое уже не получится - вы увидите ошибку:

const user **=** {

name: 'Иван'

};

user **=** 5; //нельзя менять, будет ошибка

**Деструктуризация**

Следующее понятие, которое появилось в ES6 называется *деструктуризация*. Деструктуризация - это разделение массива или объекта в отдельные переменные.

Пусть у нас дан массив ['Иванов', 'Иван']. Давайте фамилию положим в переменную **surname**, а имя - в переменную **name**. Для этого массив присвоим вот такой конструкции: **var [surname, name]**, вот так: **var [surname, name] = ['Иванов', 'Иван']**.

Эта конструкция [surname, name] и есть деструктуризация. Получится, что первый элемент массива (то есть 'Иванов') запишется в переменную **surname**, а второй - в переменную **name**.

Давайте посмотрим на примере:

let [surname, name] **=** ['Иванов', 'Иван'];

alert(surname); //выведет 'Иванов'

alert(name); //выведет 'Иван'

Можно начать записывать в переменные не сначала массива, а пропустить некоторые значения. Давайте, к примеру, пропустим фамилию, а имя и возраст запишем в переменные. Для этого при указании переменных перед первой переменной поставим запятую, вот так: [, name, age].

Посмотрим на примере:

let [, name, age] **=** ['Иванов', 'Иван', '20 лет'];

alert(name); //выведет 'Иван'

alert(age); //выведет '20 лет'

Можно пропустить не одно значение, а несколько:

let [,, age] **=** ['Иванов', 'Иван', '20 лет'];

alert(age); //выведет '20 лет'

Если в массиве больше элементов, чем переменных - лишние элементы никуда не запишутся и ничего страшного не произойдет:

let [surname, name] **=** ['Иванов', 'Иван', 20, 'женат', 'без вп'];

alert(surname); //выведет 'Иванов'

alert(name); //выведет 'Иван'

Можно сохранить оставшиеся значения массива в отдельный массив. Для этого перед переменной, в которую положится остаток, следует написать **троеточие**. В следующем примере фамилия и имя запишутся в соответствующие переменные, а остаток массива в переменную **rest**:

let [surname, name, ...rest] **=** ['Иванов', 'Иван', 20, 'женат', 'без вп'];

alert(surname); //выведет 'Иванов'

alert(name); //выведет 'Иван'

alert(rest); //выведет [20, 'женат', 'без вп']

Если в массиве меньше элементов, чем переменных, то в "лишние" переменные запишется **undefined**:

let [surname, name] **=** ['Иванов'];

alert(surname);

alert(name); //undefined

Для переменных можно указывать значения по умолчанию. В этом случае, если переменной не хватит элемента массива - возьмется значение по умолчанию.

В следующем примере переменной **name** по умолчанию указано значение 'Аноним':

let [surname, name **=** 'Аноним'] **=** ['Иванов'];

alert(surname); //введет 'Иванов'

alert(name); //введет 'Аноним'

А вот если для переменной **name** будет значение в массиве - значение по умолчанию будет проигнорировано:

let [surname, name **=** 'Аноним'] **=** ['Иванов', 'Иван'];

alert(surname); //введет 'Иванов'

alert(name); //введет 'Иван'

В качестве значения по умолчанию можно также указывать функцию:

function getName() {

return 'Аноним';

}

let [surname, name **=** getName()] **=** ['Иванов'];

alert(surname); //введет 'Иванов'

alert(name); //введет 'Аноним'

Интересный пример применения деструктуризации - поменяем переменные местами:

var a **=** 1, b **=** 2;

[a, b] **=** [b, a];

**Деструктуризация объектов**

Кроме массивов можно также делать и деструктуризацию объектов. В следующем примере значения объекта разбиваются на соответствующие переменные (имена переменных должны совпадать в ключами объекта, порядок не имеет значения):

let options **=** {

color: 'red',

width: 400,

height: 500

};

let {color, width, height} **=** options;

alert(color); //выведет 'red'

alert(width); //выведет 400

alert(height); //выведет 500

Можно сделать так, чтобы имена переменных не совпадали в именами ключей объекта:

let options **=** {

color: 'red',

width: 400,

height: 500

};

let {color: c, width: w, height: h} **=** options;

alert(c); //выведет 'red'

alert(w); //выведет 400

alert(h); //выведет 500

Можно также указывать значения по умолчанию. К примеру, укажем для цвета по умолчанию значение 'black', закомментируем элемент объекта с ключом color - и теперь в переменную **color** положится 'black':

let options **=** {

//color: 'red',

width: 400,

height: 500

};

let {color **=** 'black', width, height} **=** options;

alert(color); //выведет 'black'

Можно также менять названия переменных на свои при этом задавая значения по умолчанию:

let options **=** {

//color: 'red',

width: 400,

height: 500

};

let {color:c **=** 'black', width, height} **=** options;

alert(c);

alert(width);

alert(height);

**Итераторы**

В ES6 появились так называемые *итераторы* for-of, которые позволяют более удобно перебирать элементы массива, подобно объектам через for-in:

let arr **=** [1, 2, 3];

for (let value of arr) {

alert(value); //выведет 1, 2, 3

}

Напоминаю, почему массивы нельзя перебирать через for-in - это вызовет проблемы, если с массивом поработали как с объектом и добавили в него какое-либо свойство: в этом случае это свойство также попадет в перебор (а мы этого не хотели):

var list **=** [1, 2, 3];

list.key **=** 'elem'; //поработали с массивом как с объектом

//Тут в перебор попадает 'elem':

for (var key in list) {

console.log(list[key]); //1, 2, 3, 'elem'

}

//А тут 'elem' не попадает:

for (var value of list) {

console.log(value); //1, 2, 3

}

**Итераторы для строк**

Итераторы работают и для строк - в этом случае в цикле строка будет перебираться посимвольно:

for (let symbol of 'слово') {

alert(symbol); //выведет 'c', 'л', 'о', 'в', 'о'

}

**Строки**

В JavaScipt в строках не должно быть переноса строки - это приведет к ошибке:

var str **=** '

a

b

';

alert(str); //выдаст ошибку

Это поправили в ES6 - только строки нужно брать не в обычные кавычки, а в косые:

var str **=** `

a

b

`;

alert(str); //будет работать

В косых кавычках можно вставлять выражения в формате ${имя переменной} - в этом случае туда подставятся значения переменных, а также выполнятся математические операции:

let num1 **=** 2;

let num2 **=** 3;

alert(`${num1} + ${num2} = ${num1 **+** num2}`); //выведет '2 + 3 = 5'

**Нововведения в функциях**

Достаточно много нововведений коснулось функций. Давайте с ними разбираться.

**Значения по умолчанию**

В функциях наконец-то появились значения по умолчанию. Давайте разберем их на примере: у нас есть функция **square**, которая возводит число в квадрат. Сделаем так, чтобы, если параметр функции не передан, она возводила в квадрат число **3**, а если передан - то переданное число:

function square(num **=** 3) {

return num **\*** num;

}

alert(square(2)); //выведет 4

alert(square()); //выведет 9

Значением по умолчанию также может служить результат работы другой функции:

function square(num **=** Math.round(3.1)) {

return num **\*** num;

}

alert(square());

**Деструктуризация в функциях**

В параметрах функций также доступна деструктуризация.

Давайте сделаем функцию, принимающую произвольное количество параметров. Пусть в первый параметр запишется фамилия, во второй - имя, а в третий все остальные параметры в виде массива:

function func(surname, name, ...rest) {

alert(surname); //выведет 'Иванов'

alert(name); //выведет 'Иван'

alert(rest); //выведет ['20 лет', 'женат', 'без вп']

}

func('Иванов', 'Иван', '20 лет', 'женат', 'без вп');

Функция также может принимать объект, элементы которого запишутся в разные переменные:

let options **=** {

color: 'red',

width: 400,

height: 500

};

function func({color, width, height}) {

alert(color); //выведет 'red'

alert(width); //выведет 400

alert(height); //выведет 500

}

func(options);

Также можно указывать параметры по умолчанию:

function func({color **=** 'black', width, height}) {

}

Также можно переименовывать переменные:

function func({color: c, width, height}) {

}

Ну, и можно комбинировать параметры по умолчанию и переименование:

function func({color:c **=** 'black', width, height}) {

}

**Функции через =>**

В ES6 появился упрощенный синтаксис функций через **=>**. Эта стрелка заменяет команду **function** и, если внутри функции только одна строка, - то **return** тоже не нужен - функция вернет результат выполнения этой строки.

В следующем примере вы видите функцию в новом стиле, а ниже эквивалент в старом стиле:

let func **=** x **=>** x**+**1;

alert(func(3));

let func **=** function(x) { return x **+** 1; };

alert(func(3));

Если у функции несколько параметров - их надо брать в скобки:

let func **=** (x1, x2) **=>** x1 **+** x2;

alert(func(3, 4));

let func **=** function(x1, x2) { return x1 **+** x2; };

alert(func(3, 4));

Если функция вообще без параметров - то нужны пустые круглые скобки:

let func **=** () **=>** 3 **+** 4;

alert(func());

let func **=** function() { return 3 **+** 4; };

alert(func());

Если в функции несколько строк - необходим **return**:

let func **=** () **=>** { let a **=** 3; let b **=** 4; return a **+** b; };

alert(func());

let func **=** function() { let a **=** 3; let b **=** 4; return a **+** b; };

alert(func());

Стрелочные функции удобно использовать в качестве анонимных:

let arr **=** [1, 2, 3, 4, 5];

//Старый стиль:

arr.forEach(function(value) {

alert(value);

});

//Новый стиль:

arr.forEach(value **=>** alert(value));

**Доступность this**

Давайте рассмотрим некоторую проблему с **this**. Вы уже могли с ней сталкиваться ранее, но если вдруг не сталкивались - самое время про нее узнать.

Пусть у нас дан элемент и массив:

let arr **=** [1, 2, 3, 4, 5];

let elem **=** document.getElementById('link');

Давайте к этому элементу привяжем функцию, а в этой вызовем другую функцию, например, через **forEach**:

elem.onclick **=** function() {

arr.forEach(function(value) {

});

}

Все переменные, определенные выше **forEach** - доступны и в нем:

elem.onclick **=** function() {

var test **=** 'aaa'; arr.forEach(function(value) {

alert(test); //выведет 'aaa' в цикле

});

}

Однако, это не касается **this** - снаружи **forEach** он ссылается на наш элемент, а вот внутри него **this** будет недоступен или будет ссылаться на объект **window** (так устроен this - его значение зависит от контекста):

elem.onclick **=** function() {

//тут доступен this

arr.forEach(function(value) {

//тут this не доступен

});

}

Чтобы сделать **this** доступным внутри вложенной функции обычно поступают так: записывают его в какую-либо переменную снаружи вложенной функции (переменную можно назвать как угодно, обычно это **that** или **self**). Получается, что внутри вложенной функции наш **this** будет доступен как **that**:

elem.onclick **=** function() {

var that **=** this;

arr.forEach(function(value) {

//наш this доступен как that

});

}

В вот в стрелочных функциях нет таких проблем нет - контекст выполнения не меняется и **this** без всяких ухищрений доступен и внутри функции:

link.onclick **=** function() {

arr.forEach(elem **=>** alert(this.innerHTML **+** elem));

}

**Порядок выполнения работы**

**Практическая часть**

**Задачи для решения**

**На деструктуризацию**

1. Дан массив. Запишите первый элемент этого массива в переменную elem1, второй - в переменную elem2, третий - в переменную elem3, а все остальные элементы массива - в переменную arr.
2. Дан массив. Запишите последний элемент этого массива в переменную elem1, а предпоследний - в переменную elem2.
3. Дан массив. Запишите второй элемент этого массива в переменную elem2. Первый элемент никуда записывать не надо.
4. Дан массив. Запишите второй элемент этого массива в переменную elem2, третий - в переменную elem3. Если в массиве нет третьего элемента - запишите в переменную elem3 значение 'eee', а если нет второго - в переменную elem2 запишите значение 'bbb'. Первый элемент никуда записывать не надо.
5. Дан объект {name: 'Петр', 'surname': 'Петров', 'age': '20 лет'}. Запишите соответствующие значения в переменные name, surname и age.
6. Дан объект {name: 'Петр', 'surname': 'Петров', 'age': '20 лет', }. Запишите соответствующие значения в переменные name, surname и age. Сделайте так, чтобы, если какое-то значение не задано - оно принимало следующее значение по умолчанию: {name: 'Аноном', 'surname': 'Анонимович', 'age': '? лет'}.

**На итераторы**

1. Дан массив, выведите его элементы последовательно на экран.
2. Дан массив, выведите его элементы последовательно на экран в обратном порядке (подсказка: сначала перевернем массив через reverse).
3. Дан массив с числами, найдите сумму элементов этого массива.
4. Дана строка. С помощью for-of подсчитайте количество букв 'о' в ней.

**На функции**

1. Сделайте функцию, которая получает имя пользователя и выводит на экран 'Привет, Имя' (вместо Имя - полученное параметром имя). По умолчанию (то есть если не передать параметр) функция должна выводить 'Привет, Аноним'.
2. Дан объект с настройками, например, {id: 'elem', color: 'blue', border: '1px solid red', font: '15px Arial'}. Сделайте функцию, которая получает этот объект, извлекает из него все настройки в соответствующие переменные и для элемента с указанным id (в нашем случае это 'elem') ставит заданные CSS свойства. В объекте могут быть только эти ключи, однако, какой-либо ключ (кроме id) может быть не задан - в этом случае пусть возьмутся следующие значения по умолчанию: color: 'blue', border: '1px solid red', font: '15px Arial'.
3. Даны абзацы с числами. Сделайте так, чтобы по нажатию на абзац начинал тикать таймер, который каждую секунду будет увеличивать на единицу число в этом абзаце. По нажатию на другой абзац должно происходить то же самое - он тоже начнет тикать.