**Лабораторная работа №7**

**Тема работы:** Применение функций. Область видимости переменных

**Цель работы:** Закрепить навыки по работе и созданию пользовательских функций

**Теоретические сведения**

*Новички чащ*е *всего пытаются решить задачу просто кучей кода. Однако, это не совсем удобно. Удобнее всего выносить кусочки кода в отдельные функции. В этом случае вы получить набор простых функций, каждая из которых выполняет одно простое действие. Такие функции легко сделать, легко потестировать, легко поддерживать в дальнейшем. Также отдельные функции легко можно использовать для других задач и переносить между проектами. Давайте разбираться на задачах. Итак, задача.*

**Задача**

**Задача:** пусть *дан массив с числами, давайте запишем в новый массив только те элементы, сумма цифр в которых от 1 до 9.*

Я начну решение с разработки вспомогательной функции **getDigits** (digit - цифра), которая параметром будет принимать число, а возвращать массив цифр. Вот эта функция:

function getDigits(num) {

var str **=** String(num);

return str.split('');

}

Обратите внимание на **String(num)** - переданное число необходимо преобразовать в строку с помощью String, иначе к нему нельзя будет применить метод **split**.

Этот код можно переписать короче в одну строчку:

function getDigits(num) {

return String(num).split('');

}

Давайте потестируем нашу функцию на каком-нибудь числе:

console.log(getDigits(123)); //выведет ['1', '2', '3']

Обратите внимание на то, что функция вернула массив строк ['1', '2', '3'], а не массив чисел [1, 2, 3] - это результат работы **split**. Давайте здесь с этим ничего не будем делать, а просто учтем в дальнейшем.

Следующая функция, которую я хотел бы написать - это функция **arraySum**, параметром принимающая массив и возвращающая сумму элементов этого массива. Пусть эта функция работает и с массивом ['1', '2', '3'], и с массивом [1, 2, 3]. Это достигается с помощью функции **Number**, в которую мы оборачиваем элементы нашего массива:

function arraySum(arr) {

var sum **=** 0;

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

sum **+=** Number(arr[i]);

}

return sum;

}

Давайте потестируем комбинацию наших двух функций - найдем сумму цифр заданного числа:

var sum **=** arraySum(getDigits(123));

alert(sum); //выведет 6

Следующая функция, которую мне хотелось бы иметь - это функция **inRange**, которая параметром принимает число и возвращает true, если сумма цифр этого числа от **1** до **9**, и false - если это не так. Вот эта функция:

function inRange(num) {

var sum **=** arraySum(getDigits(num)); //используем вспомогательные функции

if (sum **>=** 1 **&&** sum **<=** 9) {

return true;

} else {

return false;

}

}

Можно переписать и короче:

function inRange(num) {

var sum **=** arraySum(getDigits(num));

return sum **>=** 1 **&&** sum **<=** 9;

}

Давайте теперь решим нашу исходную задачу. Напомню ее: дан массив с числами, запишем в новый массив только те элементы, сумма цифр в которых от 1 до 9. Для этого возьмем массив с числами, переберем его циклом и в цикле с помощью нашей функции **inRange** будем проверять, подходит ли нам этот элемент массива. Если подходит - забираем его себе:

var arr **=** [12, 19, 28, 13, 14, 345];

var result **=** [];

for (var i **=** 0; i **<** 10; i**++**) {

if (inRange(arr[i])) { //если подходит - берем

result.push(arr[i]);

}

}

console.log(result);

Окончательное решение будет выглядеть так:

var arr **=** [12, 19, 28, 13, 14, 345];

var result **=** [];

for (var i **=** 0; i **<** 10; i**++**) {

if (inRange(arr[i])) {

result.push(arr[i]);

}

}

console.log(result);

function inRange(num) {

var sum **=** arraySum(getDigits(num));

return sum **>=** 1 **&&** sum **<=** 9;

}

function getDigits(num) {

return String(num).split('');

}

function arraySum(arr) {

var sum **=** 0;

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

sum **+=** Number(arr[i]);

}

return sum;

}

**Задача**

Давайте решим еще одну задачу: *дан массив с числами, найдем сумму всех цифр из этих чисел.*

Для решения этой задачей удобно воспользоваться функциями **arraySum** и **getDigits**, которые мы разработали при решении предыдущей задачи:

var arr **=** [12, 19, 28, 13, 14, 345];

var sum **=** 0;

for (var i **=** 0; i **<** 10; i**++**) {

sum **+=** arraySum(getDigits(num));

}

alert(sum);

**Советы по работе с функциями**

**Давайте функциям осмысленные имена**

Старайтесь давать функциям осмысленные имена. Имя должно отражать то, что делает функция. Предположим, у нас есть функция, которая осуществляет валидацию (проверку правильности заполнения) формы.

Как лучше всего ее назвать? Как-нибудь так: **validateForm**. И напротив, имя **validate** или **form** не будет полностью отражать то, что делает эта функция.

В идеале вы должны посмотреть на имя функции и сразу определить то, что она делает, не заглядывая в ее код.

**Не делайте их очень большими**

Функции не должны быть очень большими. **Нормальный размер** - строк 10. Максимальный - 30 строк.

Если в вашей функции получается больше 30 - задумайтесь о том, чтобы разбить ее на несколько функций.

### Одна функция - одна задача

Функции не должны быть многозадачными. Правило такое: одна функция должна выполнять только одну задачу.

Предположим, перед вами стоит задача: при регистрации пользователя проверить, не занято ли такое имя, и если не занято - сохранить его в базу данных. Велик соблазн сделать все одной функцией, которая будет проверять, а потом сохранять.

**Однако это неправильно.** Лучше будет сделать две функции: одна проверяет, а вторая сохраняет.

**Используйте функции внутри функций**

Это правило вытекает из предыдущего - внутри функций можно и нужно использовать другие функции, вспомогательные.

**Делайте ваши функции универсальными**

Старайтесь делать ваши функции как можно более **универсальными**. Тот проект, который вы делаете сейчас - не последний в вашей жизни и написанные функции вы наверняка будете использовать в будущем. Поэтому при создании новой функции выделите пару минут и хорошо подумайте над ее именем, параметрами и их порядком (с точки зрения удобства).

**Порядок выполнения работы**

**Практическая часть**

**Задача № 1**

**Задача.** Дан массив с числами. Создайте из него новый массив, где останутся лежать только положительные числа. Создайте для этого вспомогательную функцию **isPositive()**, которая параметром будет принимать число и возвращать true, если число положительное, и false - если отрицательное.

**Решение:**

var arr **=** [1, 2, 3, **-**1, **-**2, **-**3];

function isPositive(num) {

if (num **>=**0) {

return true;

} else {

return false;

}

}

var newArr **=** [];

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

if (isPositive(arr[i])) {

newArr.push(arr[i]);

}

}

console.log(newArr);

**Задачи для решения**

1. Сделайте функцию **isNumberInRange**, которая параметром принимает число и проверяет, что оно больше нуля и меньше 10. Если это так - пусть функция возвращает **true**, если не так - **false**.
2. Дан массив с числами. Запишите в новый массив только те числа, которые больше нуля и меньше 10-ти. Для этого используйте вспомогательную функцию **isNumberInRange** из предыдущей задачи.
3. Сделайте функцию **getDigitsSum** (digit - это цифра), которая параметром принимает целое число и возвращает сумму его цифр.
4. Найдите все года от 1 до 2020, сумма цифр которых равна **13**. Для этого используйте вспомогательную функцию **getDigitsSum** из предыдущей задачи.
5. Сделайте функцию **isEven()** (even - это четный), которая параметром принимает целое число и проверяет: четное оно или нет. Если четное - пусть функция возвращает **true**, если нечетное - **false**.
6. Дан массив с целыми числами. Создайте из него новый массив, где останутся лежать только четные из этих чисел. Для этого используйте вспомогательную функцию **isEven** из предыдущей задачи.
7. Сделайте функцию **getDivisors**, которая параметром принимает число и возвращает массив его делителей (чисел, на которое делится данное число).