**Лабораторная работа №5**

**Тема работы:** Работа с функциями, флаги.

**Цель работы:** изучить способы работы с функциями и использование флагов.

**Теоретические сведения**

Функции представляют собой набор инструкций, выполняющих определенное действие или вычисляющих определенное значение.

Синтаксис определения функции:

| 1  2  3  4 | function имя\_функции([параметр [, ...]]){    // Инструкции  } |
| --- | --- |

Определение функции начинается с ключевого слова **function**, после которого следует имя функции. Наименование функции подчиняется тем же правилам, что и наименование переменной: оно может содержать только цифры, буквы, символы подчеркивания и доллара ($) и должно начинаться с буквы, символа подчеркивания или доллара.

После имени функции в скобках идет перечисление параметров. Даже если параметров у функции нет, то просто идут пустые скобки. Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

Определим простейшую функцию:

| 1  2  3  4 | function display(){    document.write("функция в JavaScript");  } |
| --- | --- |

Данная функция называется display(). Она не принимает никаких параметров и все, что она делает, это пишет на веб-страницу строку.

Однако простого определения функции еще недостаточно, чтобы она заработала. На надо еще ее вызвать:

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  </head>  <body>  <script>  function display(){    document.write("функция в JavaScript");  }  display();  </script>  </body>  </html> |
| --- | --- |

Необязательно давать функциям определенное имя. Можно использовать анонимные функции:

| 1  2  3  4  5 | var display = function(){ // определение функции    document.write("функция в JavaScript");  }  display(); |
| --- | --- |

Фактически мы определяем переменную display и присваиваем ей ссылку на функцию. А затем по имени переменной функция вызывается.

Также мы можем динамически присваивать функции для переменной:

| 1  2  3  4  5  6  7  8  9  10  11  12 | function goodMorning(){    document.write("Доброе утро");  }  function goodEvening(){    document.write("Добрый вечер");  }  var message = goodMorning;  message(); // Доброе утро  message = goodEvening;  message(); // Добрый вечер |
| --- | --- |

**Параметры функции**

Рассмотрим передачу параметров:

| 1  2  3  4  5  6  7 | function display(x){ // определение функции    var z = x \* x;  document.write(x + " в квадрате равно " + z);  }    display(5); // вызов функции |
| --- | --- |

Функция display принимает один параметр - x. Поэтому при вызове функции мы можем передать для него значение, например, число 5, как в данном случае.

Если функция принимает несколько параметров, то с помощью spread-оператора ... мы можем передать набор значений для этих параметров из массива:

| 1  2  3  4  5  6  7 | function sum(a, b, c){  let d = a + b + c;  console.log(d);  }  sum(1, 2, 3);  let nums = [4, 5, 6];  sum(...nums); |
| --- | --- |

Во втором случае в функцию передается числа из массива nums. Но чтобы передавался не просто массив, как одно значение, а именно числа из этого массива, применяется spread-оператор (многоточие ...).

**Необязательные параметры**

Функция может принимать множество параметров, но при этом часть или все параметры могут быть необязательными. Если для параметров не передается значение, то по умолчанию они имеют значение "undefined".

| 1  2  3  4  5  6  7  8  9  10 | function display(x, y){    if(y === undefined) y = 5;  if(x === undefined) x = 8;  let z = x \* y;  console.log(z);  }  display(); // 40  display(6); // 30  display(6, 4) // 24 |
| --- | --- |

Здесь функция display принимает два параметра. При вызове функции мы можем проверить их значения. При этом, вызывая функцию, необязательно передавать для этих параметров значения. Для проверки наличия значения параметров используется сравнение со значением undefined.

Есть и другой способ определения значения для параметров по умолчанию:

| 1  2  3  4  5  6  7 | function display(x = 5, y = 10){  let z = x \* y;  console.log(z);  }  display(); // 50  display(6); // 60  display(6, 4) // 24 |
| --- | --- |

Если параметрам x и y не передаются значения, то они получаются в качестве значений числа 5 и 10 соответствено. Такой способ более лаконичен и интуитивен, чем сравнение с undefined.

При этом значение параметра по умолчанию может быть производным, представлять выражение:

| 1  2  3  4  5  6  7 | function display(x = 5, y = 10 + x){  let z = x \* y;  console.log(z);  }  display(); // 75  display(6); // 96  display(6, 4) // 24 |
| --- | --- |

В данном случае значение параметра y зависит от значения x.

При необходимости мы можем получить все переданные параметры через глобально доступный массив **arguments**:

| 1  2  3  4  5  6  7  8  9 | function display(){  var z = 1;  for(var i=0; i<arguments.length; i++)  z \*= arguments[i];  console.log(z);  }  display(6); // 6  display(6, 4) // 24  display(6, 4, 5) // 120 |
| --- | --- |

При этом даже не важно, что при определении функции мы не указали никаких параметров, мы все равно можем их передать и получить их значения через массив arguments.

**Неопределенное количество параметров**

С помощью spread-оператора мы можем указать, что с помощью параметра можно передать переменное количество значений:

| 1  2  3  4  5  6  7  8 | function display(season, ...temps){  console.log(season);  for(index in temps){  console.log(temps[index]);  }  }  display("Весна", -2, -3, 4, 2, 5);  display("Лето", 20, 23, 31); |
| --- | --- |

В данном случае второй параметр ...temps указывает, что вместо него можно передать разное количество значений. В самой функции temps фактически представляет массив переданных значений, которые мы можем получить. При этом несмотря на это, при вызове функции в нее передается не массив, а именно отдельные значения.

Консольный вывод:

Весна

-2

-3

4

2

5

Лето

20

23

31

**Результат функции**

Функция может возвращать результат. Для этого используется оператор **return**:

| 1  2  3  4  5  6  7 | var y = 5;  var z = square(y);  document.write(y + " в квадрате равно " + z);    function square(x) {  return x \* x;  } |
| --- | --- |

После оператора return идет значение, которое надо возвратить из метода. В данном случае это квадрат числа х.

После получения результата функции мы можем присвоить его какой-либо другой переменной:

| 1 | var z = square(y); |
| --- | --- |

**Функции в качестве параметров**

Функции могут выступать в качестве параметров других функций:

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | function sum(x, y){  return x + y;  }    function subtract(x, y){  return x - y;  }    function operation(x, y, func){    var result = func(x, y);  console.log(result);  }    console.log("Sum");  operation(10, 6, sum); // 16    console.log("Subtract");  operation(10, 6, subtract); // 4 |
| --- | --- |

Функция operation принимает три параметра: x, y и func. func - представляет функцию, причем на момент определения operation не важно, что это будет за функция. Единственное, что известно, что функция func может принимать два параметра и возвращать значение, которое затем отображается в консоли браузера. Поэтому мы можем определить различные функции (например, функции sum и subtract в данном случае) и передавать их в вызов функции operation.

**Возвращение функции из функции**

Одна функция может возвращать другую функцию:

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | function menu(n){    if(n==1) return function(x, y){ return x+y;}  else if(n==2) return function(x, y){ return x - y;}  else if(n==3) return function(x, y){ return x \* y;}  return undefined;  }    for(var i=1; i < 5; i++){  var action = menu(i);  if(action!==undefined){  var result = action(5, 4);  console.log(result);  }  } |
| --- | --- |

В данном случае функция menu в зависимости от переданного в нее значения возвращает одну из трех функций или undefined.

**Флаги**

Давайте решим следующую задачу: дан массив, нужно проверить, есть ли в нем элемент со значением 'c' или нет. Если есть - выведем 'есть', если нет - выведем 'нет'.

Для начала давайте попробуем вывести 'есть'. Для этого переберем все элементы нашего массива и ифом будем спрашивать - равен ли текущий элемент значению 'c'. Если равен - выведем 'есть':

var arr **=** ['a', 'b', 'c', 'd', 'e'];

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

if (arr[i] **==** 'c') {

alert('Есть');

}

}

Но наше решение не очень хорошо: ведь если в массиве будет не одно значение 'c', а несколько, то алерт выведется несколько раз. Давайте исправим наш массив (сделаем два элемента с 'c') и убедимся в этом:

var arr **=** ['a', 'b', 'c', 'd', 'с'];

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

if (arr[i] **==** 'c') {

alert('Есть'); //выведет несколько раз

}

}

Поправим проблему: оборвем цикл с помощью **break**, если элемент уже найден:

var arr **=** ['a', 'b', 'c', 'd', 'с'];

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

if (arr[i] **==** 'c') {

alert('Есть');

break; //оборвем цикл

}

}

Давайте теперь попробуем сделать так, чтобы, если в массиве вообще нет элементов со значением 'c', выводилось 'нет'. Распространенным заблуждением будет добавить **else** к нашему ифу - в этом случае 'нет' будет выводиться на все элементы, не являющиеся 'c':

var arr **=** ['a', 'b', 'c', 'd', 'с'];

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

if (arr[i] **==** 'c') {

alert('Есть');

} else {

alert('Нет'); //выведет на 'a', 'b' и 'd'

}

}

Итак, идея добавить else - плохая идея, не рабочая. Для решения задач подобного типа (такие задачи встречаются достаточно часто) используют так называемые *флаги*.

**Флаг** - это такая переменная, которая может принимать два значения: true или false. Как работать с флагами - посмотрим на примере нашей задачи.

Итак, давайте сделаем переменную **flag** с таким значением: если она равна true - то в массиве есть элемент 'c', а если false - то такого элемента нет.

Изначально поставим переменную **flag** в значение false - то есть будем считать, что элемента 'c' в массиве нет:

var arr **=** ['a', 'b', 'c', 'd', 'с'];

var flag **=** false; //считаем, что элемента 'c' нету в массиве

Затем запустим цикл с ифом так, как мы делали это раньше. Если цикл обнаружит, что в массиве есть элемент 'c' - то поставим переменную **flag** в значение true (и выйдем из цикла с помощью break).

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

if (arr[i] **==** 'c') {

flag **=** true; //элемент есть - переопределим переменную flag

break;

}

}

Далее: ответ на вопрос, если в массиве 'c' или нет, мы можем дать только после цикла. И этот ответ у нас уже есть: после цикла переменная **flag** могла остаться false или могла сменить свое значение на true, если цикл обнаружил в массиве 'c':

var arr **=** ['a', 'b', 'c', 'd', 'с'];

var flag **=** false; //считаем, что элемента 'c' нету в массиве

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

if (arr[i] **==** 'c') {

flag **=** true; //элемент есть - переопределим переменную flag

break;

}

}

//тут переменная flag равна или true, или false

Теперь после цикла мы можем сделать иф, который посмотрит на переменную flag и выведет на экран 'есть' или 'нет':

var arr **=** ['a', 'b', 'c', 'd', 'с'];

var flag **=** false; //считаем, что элемента 'c' нету в массиве

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

if (arr[i] **==** 'c') {

flag **=** true; //элемент есть - переопределим переменную flag

break;

}

}

if (flag **===** true) {

alert('Есть');

} else {

alert('Нет');

}

Таким образом и работают флаги. Запомните этот прием - он обязательно встретится вам в дальнейшем, причем неоднократно.

**Флаги в функции**

Когда мы работаем с пользовательскими функциями - флаги тоже можно и нужно использовать, но работа с ними упрощается. Давайте решим ту же задачу на 'c', но таким образом: необходимо создать функцию **hasElem**, которая параметром будет принимать массив и возвращать true, если 'c' есть в массиве, и false - если нет:

var arr **=** ['a', 'b', 'c', 'd', 'с'];

alert(hasElem(arr)); //выведет true или false

Давайте реализуем функцию **hasElem**. В этом случае переменная flag и команда break нам не понадобится - вместо них все сделает команда return.

Как это будет: внутри функции запускаем наш цикл с ифом и, если мы попали в иф - делаем **return** true, тем самым выходя из цикла и из функции. А если цикл прокрутился, но выхода из функции не случилось - после цикла делаем **return false**:

function hasElem(arr) {

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

if (arr[i] **==** 'c') {

return true;

}

}

return false; //элемент 'c' не найден

}

var arr **=** ['a', 'b', 'c', 'd', 'с'];

alert(hasElem(arr));

Еще раз как это работает: если в массиве найдется элемент со значением 'c' - мы выйдем из функции с помощью return. Но если в массиве не найдется элемент 'c' - выхода из функции не произойдет и выполнение дойдет до команды return false. И получится, что функция вернет false в знак того, что элемент 'c' не найден в массиве.

**Порядок выполнения работы**

**Примеры решения задач**

### Задача №1

**Задача.** Сделайте функцию, которая возвращает куб числа. Число передается параметром.

**Решение:**

function cube($num) {

return $num **\*** $num **\*** $num

}

**Практическая часть**

**Простые** **функции**

1. Сделайте функцию, которая возвращает квадрат числа. Число передается параметром.
2. Сделайте функцию, которая возвращает сумму двух чисел.
3. Сделайте функцию, которая отнимает от первого числа второе и делит на третье.
4. Сделайте функцию, которая принимает параметром число от **1** до **7**, а возвращает день недели на русском языке.

**На флаги**

1. Дан массив с числами. Проверьте, что в этом массиве есть число **5**. Если есть - выведите 'да', а если нет - выведите 'нет'.
2. Дано число, например **31**. Проверьте, что это число не делится ни на одно другое число кроме себя самого и единицы. То есть в нашем случае нужно проверить, что число 31 не делится на все числа от 2 до 30. Если число не делится - выведите 'false', а если делится - выведите 'true'.
3. Дан массив с числами. Проверьте, есть ли в нем два одинаковых числа подряд. Если есть - выведите 'да', а если нет - выведите 'нет'.

**Контрольные вопросы**

1. Что такое функции?
2. Для чего нужен **spread-**оператор?
3. Как создать анонимную функцию?
4. Что такое флаги?