

Функции / Functions

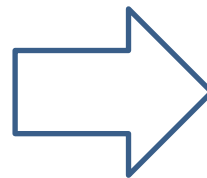


JavaScript
Courses

vk.com/js.courses

js.courses.dp.ua/files

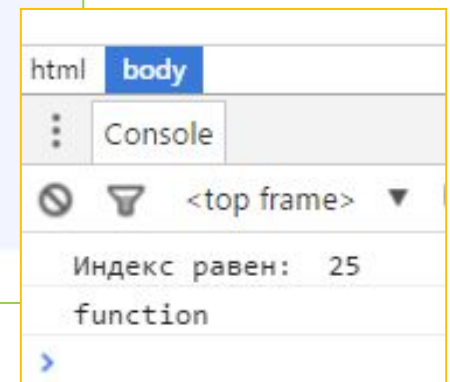
**Функции – блок кода, который пишется
один раз, а потом может
использоваться многократно
(процедуры, подпрограммы).**



Идея функций заключается в следующем: зачем писать многократно одно и то же, лучше сказать программе: я уже такое писал, возьми и повтори здесь, там, и еще вот там.

ФУНКЦИИ В JavaScript

```
1 <script>
2   function imt(height, weight){
3       height = height / 100;
4       var result = weight / (height*height);
5       return result;
6   }
7
8   /*
9   var imt = function(height, weight){
10      height = height / 100;
11      var result = weight / (height*height);
12      return result;
13   }
14   */
15
16   console.log("Индекс: " + imt(200, 100));
17   console.log(typeof(imt));
18 </script>
```



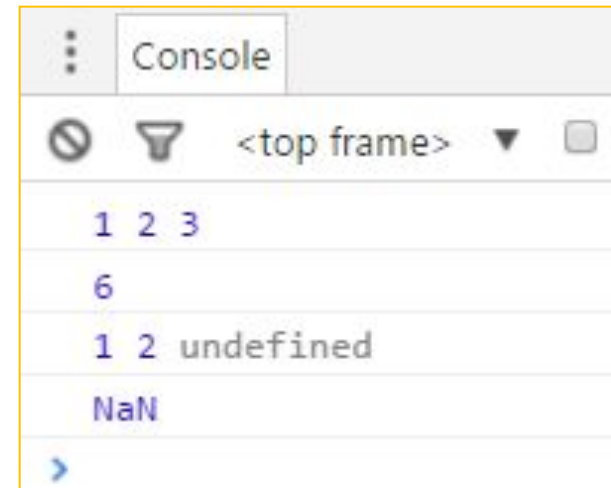
Функции в

JavaScript

Параметры функции (аргументы) – такие себе «переменные» которые передаются функции при каждом вызове, и могут влиять на результат её работы. Существуют только в теле функции.

Возвращаемое значение – возможность функции вернуть результат своей работы, при необходимости, при помощи оператора **return**.

```
1 <script>
2   function demo(a, b, c)
3   {
4       console.log(a, b, c);
5       return a+b+c;
6   }
7
8   var result = demo(1, 2, 3);
9   console.log(result);
10  //-----//
11  result = demo(1, 2);
12  console.log(result);
13
14 </script>
```



Функции в

JavaScript
Параметры функции (аргументы) – такие себе «переменные» которые передаются функции при каждом вызове, и могут влиять на результат её работы. Существуют только в теле функции. Не влияют на внешние переменные простых типов которые были переданы функции.

```
1 <script>
2   function test(a){
3     console.log(a);
4     a = 7;
5     console.log(a);
6   }
7
8   var z = 3;
9
10  test(z);
11
12  console.log(z);
13 </script>
```

All	Errors	Warnings	Info	Logs	Debug	Handled
				3		xxx.html:4
				7		xxx.html:6
				3		xxx.html:13
				>		

Практическая ценность функций

```
1 <!DOCTYPE html>
2 <html><head><title>Основы JavaScript</title><meta charset="utf-8" />
3 <script>
4   function tag_appender(tag_text, tag_name, tag_color){
5     var new_tag      = document.createElement(tag_name);
6     new_tag.innerHTML = tag_text;
7     new_tag.style.color = tag_color;
8     document.body.appendChild(new_tag);
9   };
10
11  window.onload = function(){
12    tag_appender("Текст_123", "p", "green");
13    tag_appender("ЛЯЛЯЛЯЛЯЛЯЛЯЛЛ", "div", "purple");
14    tag_appender("---", "i", "red");
15  }
16
17 </script>
18 </head><body></body></html>
```

Меньше кода, многократное использование кода, в случае изменений: изменение вносится один раз, результат будет во всех местах где используется функция.

Практическая ценность функций

The screenshot shows a web browser window with the following content:

Текст_123
ЛЯЛЯЛЯЛЯЛЯЛЛ

The browser's developer tools are open, showing the following HTML structure:

```
<!DOCTYPE html>  
<html>  
  <head>...</head>  
  <body>  
    <p style="color: green;">Текст_123</p>  
    <div style="color: purple;">ЛЯЛЯЛЯЛЯЛЯЛЛ</div>  
    <i style="color: red;">---</i>  
  </body>  
</html>
```

The Styles pane on the right shows the following CSS rules:

```
element.style {  
}  
body {  
  user agent stylesheet  
  display: block;  
  margin: 8px;  
}
```

The diagram below the styles pane illustrates the box model for the selected element:

- margin: 8px (orange dashed border)
- border: - (yellow solid border)
- padding: - (green solid border)
- Content area: 658 x 70 (blue solid area)

The browser's address bar shows the file path: file:///T:/SHARE_WORK/Dropbox/WEB_DEV_COURSES_April_2015/

ФУНКЦИИ В

JavaScript

```
1 <!DOCTYPE html>
2 <html><head>
3 <script>
4   function a(){
5     console.log("test_1");
6   };
7
8   var b = function(){
9     console.log("test_2");
10  };
11
12  var c = a;
13  var d = b;
14
15  a();
16  b();
17  c();
18  d();
19
20  console.log(typeof(a), typeof(b),
21  |typeof(c), typeof(d));
22
23  var x = 23;
24  x();
25 </script>
26 </head><body></body></html>
```

test_1	example.html:5
test_2	example.html:9
test_1	example.html:5
test_2	example.html:9
function function function function	example.html:20
✖ Uncaught TypeError: x is not a function	example.html:23

ФУНКЦИИ В

JavaScript
Параметры функции — это такие себе «переменные» которые передаются функции при каждом вызове

```
1 <!DOCTYPE html>
2 <html><head><title>Основы JavaScript</title><meta charset="utf-8" />
3 <script>
4   function a(param1_name, param2_years){
5     console.log("Добрый день " + param1_name);
6     console.log("Ваш возраст (в годах): " + param2_years);
7   };
8
9   a("Вася", 33);
10
11   a("Петя");
12
13   a();
14
15   a(23, "Вася", 12, 22, true);
16
17 </script>
18 </head><body></body></html>
```

⊗ 🔍 <top frame> ▾ Preserve log

Добрый день Вася	code 2.html:5
Ваш возраст (в годах): 33	code 2.html:6
Добрый день Петя	code 2.html:5
Ваш возраст (в годах): undefined	code 2.html:6
Добрый день undefined	code 2.html:5
Ваш возраст (в годах): undefined	code 2.html:6
Добрый день 23	code 2.html:5
Ваш возраст (в годах): Вася	code 2.html:6

ФУНКЦИИ В

JavaScript – возможность функции вернуть результат своей работы, при необходимости.

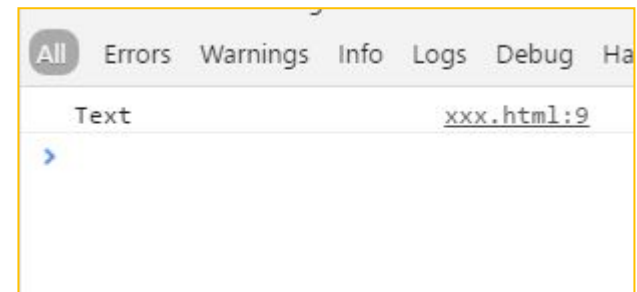
```
1 <!DOCTYPE html>
2 <html><head><title>Основы JavaScript</title><meta charset="utf-8" />
3 <script>
4     function my_func(a, b){
5         return a + b;
6     };
7
8     console.log( my_func(3,4) );
9     console.log( my_func(" text ",4) );
10    console.log( my_func() );
11
12    function xxx(){
13        var t = 123;
14        return t;
15    }
16
17    function yyy(){
18        var t = 123;
19    }
20
21    console.log( xxx(), yyy() );
22
23 </script>
24 </head><body></body></html>
```

Message	Source
7	code 3.html:8
text 4	code 3.html:9
NaN	code 3.html:10
123 undefined	code 3.html:21

ФУНКЦИИ В JavaScript

Оператор *return*

```
1 <script>
2
3   function test(a) {
4
5       if(a > 7) {
6           return;
7       };
8
9       console.log("Text");
10  }
11
12  test(8);
13
14  test(5);
15
16 </script>
```



Оператор *return* помимо возврата значения из функции еще имеет свойство прервать выполнение функции в любой момент. Этим он похож на оператор *break* для циклов.

Переменные в теле функции, локальные переменные, область

```
1 <script>
2
3   function test () {
4       var sms = "Hi all!";
5       console.log (sms) ;
6   }
7
8   test () ;
9
10  console.log (sms) ;
11
12 </script>
```

Hi all!

[exmaple.html:5](#)

Uncaught ReferenceError: sms is not defined

[exmaple.html:10](#)

Переменные объявленные в теле функции называют локальными, такие переменные существуют только внутри тела функции. Они создаются каждый раз при начале работы функции и уничтожаются при завершении её работы.

Глобальные переменные

```
1 <script>
2
3   var a = 6;
4
5   function test () {
6     a++;
7   }
8
9   console.log(a);
10
11  test();
12
13  console.log(a);
14
15 </script>
```

6	<u>example.html:9</u>
7	<u>example.html:13</u>

Глобальные переменные – те которые объявлены вне тела функции, функции имеют доступ к ним, и могут их необратимо изменять.

Соккрытие глобальных переменных

```
1 <script>
2
3   var a = 7;
4
5   function test() {
6     var a = 8;
7   }
8
9   console.log(a);
10
11  test();
12
13  console.log(a);
14
15 </script>
```

7	<u>exmaple.html:9</u>
7	<u>exmaple.html:13</u>

Повторное объявление глобальной переменной в теле функции сокращает глобальную переменную, и функция будет работать только с локальной заменой, которая будет уничтожена после завершения вызова функции.

Области видимости переменных

Всё что касается глобальных и локальных переменных относится только к функциям. На другие средства выделения блока кода (if, for и т. д.) эти правило не распространяются.

Применение функций. Таймеры

```
1 <script>
2
3   var test_interval = function() {
4       var now = new Date();
5       console.log("Function 'test_interval': " + now.getHours() + ":" +
6           now.getMinutes() + ":" + now.getSeconds() + "." + now.getMilliseconds());
7   }
8
9   var test_timeout = function() {
10      var now = new Date();
11      console.log("Function 'test_timeout': " + now.getHours() + ":" +
12          now.getMinutes() + ":" + now.getSeconds() + "." + now.getMilliseconds());
13  }
14
15  setTimeout(test_timeout, 3000);
16
17  setInterval(test_interval, 5000);
18
19  var now = new Date();
20  console.log("Script start: " + now.getHours() + ":" + now.getMinutes() + ":" +
21      now.getSeconds() + "." + now.getMilliseconds());
22 </script>
```

Script start: 22:49:15.255	example.html:19
Function 'test_timeout': 22:49:18.255	example.html:10
Function 'test_interval': 22:49:20.255	example.html:5
Function 'test_interval': 22:49:25.255	example.html:5
Function 'test_interval': 22:49:30.254	example.html:5
Function 'test_interval': 22:49:35.255	example.html:5

Применение функций. Таймеры

```
1 <script>
2
3   var test_interval = function(){
4       var now = new Date();
5       console.log("Function 'test_interval': " + now.getHours() + ":" +
6           now.getMinutes() + ":" + now.getSeconds() + "." + now.getMilliseconds());
7   }
8
9   var test_timeout = function(){
10      var now = new Date();
11      console.log("Function 'test_timeout': " + now.getHours() + ":" +
12          now.getMinutes() + ":" + now.getSeconds() + "." + now.getMilliseconds());
13  }
14
15  setTimeout(test_timeout, 3000);
16
17  setInterval(test_interval, 5000);
18
19  var now = new Date();
20
21  console.log("Script start: " + now.getHours() + ":" + now.getMinutes() + ":" +
22      now.getSeconds() + "." + now.getMilliseconds());
23 </script>
```

setTimeout(*some_function*, *delay*) – вызовет функцию *some_function* через *delay* миллисекунд. Сделает это один раз.

setInterval(*some_function*, *delay*) – вызовет функцию *some_function* через *delay* миллисекунд. И будет повторять вызов каждые *delay* миллисекунд.

Список аргументов переменной

```
1 <script>
2
3   var func = function(){
4     console.log("В этой функции " + arguments.length + " аргументов.");
5
6     for(var i in arguments){
7       console.log(i + ": " + arguments[i]);
8     }
9   };
10
11  func(1,4, "lalala");
12
13  console.log("---");
14
15  func();
16
17  console.log("---");
18
19  func(true, false);
20
21 </script>
```

В этой функции 3 аргументов.	arguments_example.html:4
0: 1	arguments_example.html:7
1: 4	arguments_example.html:7
2: lalala	arguments_example.html:7
---	arguments_example.html:12
В этой функции 0 аргументов.	arguments_example.html:4
---	arguments_example.html:14
В этой функции 2 аргументов.	arguments_example.html:4
0: true	arguments_example.html:7
1: false	arguments_example.html:7

Псевдомассив **arguments** – хранит в себе все аргументы (параметры) которые, при вызове, переданы функции, и их порядковые номера.

Список аргументов переменной

```
1 <script>
2
3 var func = function(a, b){
4     console.log("В этой функции " + arguments.length + " аргументов.");
5
6     for(var i in arguments){
7         console.log(i + ": " + arguments[i]);
8     }
9
10    console.log("a: ", a);
11    console.log("b: ", b);
12 };
13
14 func(true, 56, "Polina");
15
16 </script>
```

В этой функции 3 аргументов.	arguments_example.html:4
0: true	arguments_example.html:7
1: 56	arguments_example.html:7
2: Polina	arguments_example.html:7
a: true	arguments_example.html:10
b: 56	arguments_example.html:11

Псевдомассив *arguments* – хранит в себе все аргументы (параметры) которые, при вызове, переданы функции, и их порядковые номера. Все параметры которые предусмотрены в заголовке функции, также хранятся в *arguments*

Самовызывающаяся функция

```
1 <script>
2
3     (function() {
4         console.log("Это самовызывающаяся функция");
5     }) ();
6
7 </script>
```

Это самовызывающаяся функция

arguments_example.html:4

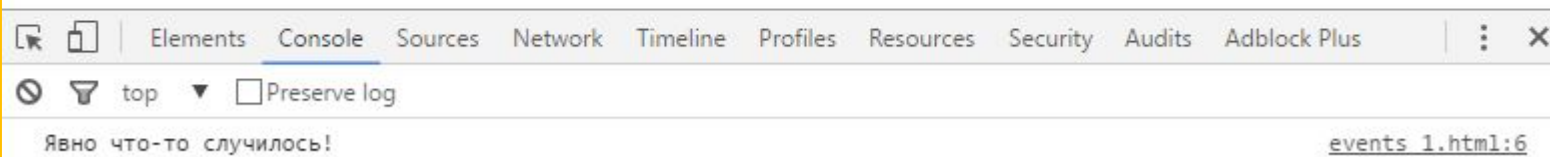
Самовызывающаяся функция – удобный механизм выполнить какие-либо действия автоматически, не создавая переменных и внося в код явных вызовов функций. Другими словами не засоряя глобальную область видимости. Активно используется в сторонних библиотеках.

Функции и события

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script>
5     function my_func() {
6       console.log("Явно что-то случилось!");
7     }
8   </script>
9 </head>
10 <body>
11   <div>This is a text</div>
12   <button onclick="my_func();">Click me!</button>
13 </body>
14 </html>
```

This is a text

Click me!



На базе функций работает система событий (оповещения о выполнении пользователем тех или иных действий)

Функции и события

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script>
5
6     window.onload = function() {
7       var div = document.querySelector("div");
8       div.onclick = my_func;
9     }
10
11   function my_func() {
12     console.log("Явно что-то случилось!");
13   }
14 </script>
15 </head>
16 <body>
17   <div>This is a text</div>
18   <button onclick="my_func();" >Click me!</button>
19 </body>
20 </html>
```

Явно что-то случилось!	events_1.html:12
Явно что-то случилось!	events_1.html:12
>	

На базе функций работает система событий (оповещения о выполнении пользователем тех или иных действий)

Функции и forEach

```
1 <script>
2
3   function zzz(elem, i, current_mas){
4       console.log(i, elem, current_mas);
5   }
6
7   var mas = ["Ivan", "Elena", "Yulia", "Marina", "Pavel"];
8
9   mas.forEach(zzz);
10
11 </script>
```

0	"Ivan" ["Ivan", "Elena", "Yulia", "Marina", "Pavel"]	forEach.html:4
1	"Elena" ["Ivan", "Elena", "Yulia", "Marina", "Pavel"]	forEach.html:4
2	"Yulia" ["Ivan", "Elena", "Yulia", "Marina", "Pavel"]	forEach.html:4
3	"Marina" ["Ivan", "Elena", "Yulia", "Marina", "Pavel"]	forEach.html:4
4	"Pavel" ["Ivan", "Elena", "Yulia", "Marina", "Pavel"]	forEach.html:4

Метод **forEach** у «классических» массивов, позволяет применить функцию, которую передают в качестве аргумента, к каждому элементу массива. Эта функция будет вызвана столько раз сколько элементов содержит массив. При каждом вызове ей будет передаваться один из элементов массива. Работает только с «классическими» массивами.

Домашнее задание

Из занятия №5

Разработать скрипт, проверяющий знания таблицы умножения двузначных чисел. Скрипт должен задать пользователю 12 задач на умножение двузначных чисел. По результатам проверки, пользователю выставляется оценка, а также выводиться два списка: верных ответов, и ошибочных ответов, указанием какой ответ был правильный.

Дополнительное условие: если ответ был дан более чем через 15 секунд – считать его не верным.

JavaScript как язык программирования

его

концепции

Переменные / Типы /

Операции

Ветвления (условные

операторы)

Циклы / Массивы (структуры

**данных)
Функции**

Объект

ы