

Функции

Определение

Функция – это фрагмент кода, который можно выполнить многократно в разных частях программы. Т.е. одни и те же действия много раз с разными исходными значениями.

Функции позволяют разбивать сложные, комплексные задачи на более мелкие части. Это повышает читабельность программы и упрощает ее переиспользование. Функции являются основными «строительными блоками» программы.

Примеры **встроенных** функций вы уже видели – это

- `alert(message)`, `prompt(message, default)` и `confirm(question)`
- `Math.sqrt()`
- `console.log()`

Можно создавать пользовательские функции.

Способы создания функций

JavaScript позволяет создавать функцию различными способами:

- **Function Declaration** (Классический способ через function);
- **Function Expression** (Функциональное Выражение);
- **Arrow Function**(Стрелочные функции).

Function Declaration

классический способ

Синтаксис объявления (создания) функции

```
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.

Имя функции

Функции, начинающиеся с...

- **"get..."** – возвращают значение,
- **"show..."** – показывает значение,
- **"calc..."** – что-то вычисляют,
- **"create..."** – что-то создают,
- **"check..."**, **"is..."** – что-то проверяют и возвращают логическое значение

Пример объявления функции:

```
function showHello() {  
    alert( 'Hello!' );  
}
```

При этом когда мы объявляем функцию с именем, мы тем самым по сути создаём новую переменную с этим названием. Эта переменная будет функцией.

Объявленная функция сама по себе не выполняется. Запуск функции выполняется посредством её вызова.

Общий синтаксис вызова функции:

Чтобы функция выполнила свою работу, нам надо ее вызвать. Для вызова функции необходимо указать её имя и две круглые скобки, в которых при необходимости ей можно передать аргументы. Отделение одного аргумента от другого выполняется с помощью запятой.

имя_функции (параметры)

Пример вызова функции:

Наш пример вызова функции. Вызов `showHello()` выполняет код функции. Отличительной чертой функций является то, что их можно многократно вызывать в различных местах программы:

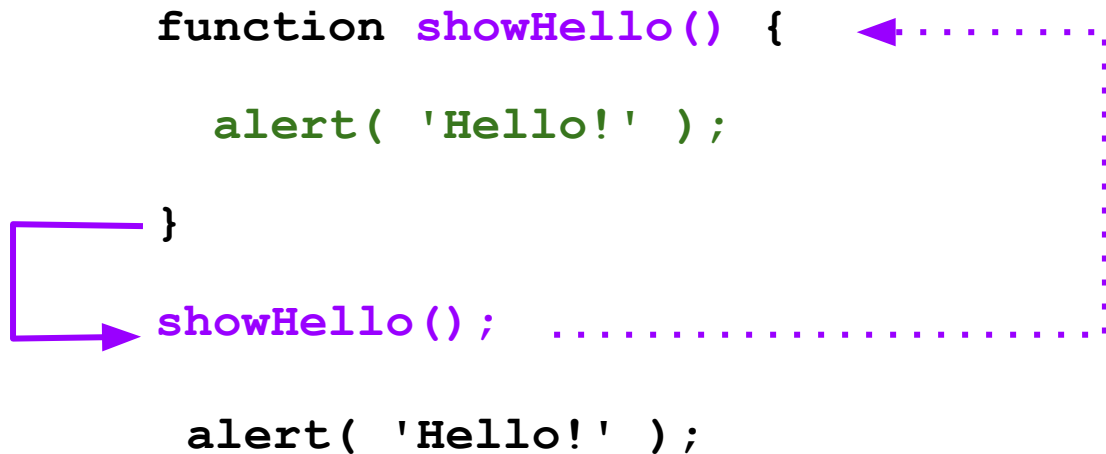
```
showHello() ;
```

```
...
```

```
showHello() ;
```

Как это работает

```
function showHello() {  
    alert( 'Hello!' );  
}  
showHello();  
  
alert( 'Hello!' );
```



1- при вызове функции идет обращение к функции по ее имени

2- происходит исполнение кода функции, программа при этом временно останавливает выполнение (при работе синхронных функций)

3- результат работы кода функции добавляется в месте ее вызова

Локальные переменные

Тело функции можно представить себе, как отдельную программу. Эта программа является достаточно независимой: в ней могут описываться собственные переменные, организовываться свои циклы, проверяться необходимые условия и т.п. — в теле функции доступны все средства программирования.

Переменные, объявленные внутри функции, видны только внутри этой функции!!!

```
function имя_функции() {  
    let переменные;  
    // Инструкции  
}
```

Пример

```
function showTxt() {  
    let txt = "Привет!"; // локальная переменная  
    alert( txt );  
}  
  
...  
  
showTxt(); // alert( Привет!);  
  
alert( txt ); //ошибка, т.к. переменная не существует
```

Глобальные (внешние) переменные

Функция обладает полным доступом к внешним переменным!!!.

```
let nameLP = "JS";
```

```
function showTxt() {
```

```
    let txt = "Привет, " + nameLP;
```

```
    alert( txt );
```

```
}
```

```
showTxt();           // alert( Привет, JS);
```

```
alert( txt );        //ошибка, т.к. переменная не существует
```

```
alert( nameLP );     //alert( JS);
```

Глобальные переменные

Функция обладает полным доступом к внешним переменным и может изменять их значение!!!.

```
let nameLP = "JS";

function showTxt() {
    nameLP = "C++"; //изменяем значение внешней переменной
    let txt = "Привет, " + nameLP;
    alert( txt );
}

alert( nameLP ); //alert(JS); перед вызовом
showTxt();      // alert( Привет, C++);
alert( nameLP ); //alert(C++); после вызова
```

Глобальные и локальные переменные

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю. Например, в коде ниже функция использует локальную переменную `nameLP`, внешняя будет проигнорирована (не изменена)

```
let nameLP = "JS";
```

```
function showTxt() {
```

```
    let nameLP = "C++"; //объявляем локальную переменную, функция создаст и будет  
использовать свою собственную локальную переменную
```

```
    let txt = "Привет, " + nameLP;
```

```
    alert( txt );
```

```
}
```

```
alert( nameLP ); //alert(JS); перед вызовом
```

```
showTxt(); // alert( Привет, C++);
```

```
alert( nameLP ); //alert(JS); после вызова, не изменилась, функция не трогала внешнюю  
переменную
```

Глобальные переменные

Глобальные переменные видимы для любой функции (если только их не перекрывают одноимённые локальные переменные).

Желательно сводить использование глобальных переменных к минимуму!

В современном коде обычно мало или совсем нет глобальных переменных. Хотя они иногда полезны для хранения важнейших «общепроектных» данных.

Параметры и аргументы

Параметры – это по сути переменные, которые описываются в круглых скобках на этапе объявления функции. Параметры доступны только внутри функции, получить доступ к ним снаружи нельзя. Значения параметры получают в момент вызова функции, т.е. посредством аргументов.

Аргументы – это значения, которые мы передаём в функцию в момент её вызова.

```
function имя_функции(параметр) {  
    // Инструкции  
}  
  
имя_функции(аргумент)
```

Пример работы функции с параметрами

```
function showTxt(nameLP) {  
    let txt = "Привет, " + nameLP;  
    alert(txt);  
}  
  
showTxt("JS"); // alert( Привет, JS);  
showTxt("C++"); // alert( Привет, C++);  
showTxt("HTML"); // alert( Привет, HTML);
```

Когда функция вызывается, переданные значения (аргументы) копируются в локальную переменную `nameLP`. Затем она используется в теле функции.

Пример работы функции с параметрами и переменными

```
function showTxt(nameLP) {  
    nameLP = nameLP + " !";  
    let txt = "Привет, "+ nameLP;  
    alert(txt);  
}  
  
let nameLP = "JS";  
  
showTxt(nameLP); // alert( Привет, JS !);  
  
showTxt("C++"); // alert( Привет, C++ !);  
  
alert(nameLP); // alert( JS );
```

Когда функция вызывается, переданные значения (аргументы) копируются в локальную переменную nameLP. Затем она используется в теле функции. **Функция всегда получает только копию значения!** То есть функция изменяет значение локальной nameLP, но не изменяет значение внешней(глобальной) переменной nameLP

Задание

- Создайте функцию `sayError()`, которая будет выводить (при помощи диалогового окна `alert`) сообщение с текстом «`Some error occurred!`».
- Создайте функцию `showName(s)`, которая будет выводить приветствие в виде «`Привет, Name!`». Имя пользователя задается через `prompt()`
- Создайте функцию `showError(x)`, которая будет выводить (при помощи диалогового окна `alert`) сообщение с текстом «`Error X occurred!`», где `X` — текст из аргумента функции
(например, вызов `showError('Out of memory')` должен вывести сообщение «`Error, Out of memory occurred!`»).

```
// объявление функции
```

```
function showMsg(name) {
```

```
    alert("Привет, " + name + "!");
```

```
}
```

```
// значение переменной name будет меняться в зависимости от введенных данных
```

```
let name = prompt("Введите имя: ");
```

```
showMsg(name); // вызов функции
```

Функция для сложения двух чисел

```
function sum(a, b) {  
    let s = a + b;  
    alert(s);  
}
```

```
// ВЫЗОВ ФУНКЦИИ
```

```
sum(1,6);    //7
```

```
sum(-1,10);  //9
```

```
sum(1);      //Nan
```

```
sum(1,2,4);  //3
```

Изменить, чтобы пользователь сам мог вводить числа

Значение параметров функции по умолчанию

```
function sum(a=0, b=0) {
```

```
    let s = a + b;
```

```
    alert(s);
```

```
}
```

```
// ВЫЗОВ ФУНКЦИИ
```

```
sum(1, 6) ; // 7
```

```
sum(1) ; // 1
```

```
sum() ; // 0
```

Возврат значения

Функция всегда возвращает значение, даже если мы не указываем это явно. По умолчанию она возвращает значение `undefined`.

Функция может вернуть результат , который будет передан в вызвавший её код.

Для возврата результата работы функции используют оператор `return`.

Оператор `return` передает управление вызывающей стороне. Это значит, что никакой код после `return` не выполняется.

Если не указывать оператор `return`, функция вернет значение `undefined`. Пустой `return` аналогичен `return undefined`

Директива `return` может находиться в любом месте тела функции. Как только выполнение доходит до этого места, функция останавливается, и значение возвращается в вызвавший её код

Функция для сложения двух чисел

```
function sum(a, b) {  
  
    let s = a + b;  
  
    return s;  
  
}  
  
// ВЫЗОВ ФУНКЦИИ  
  
let result = sum(1,6);  
  
alert("Сумма введенных чисел: " + result);
```

Пример, ограничение / разрешение доступа несколько return

Пользователь вводит возраст , и получает / не получает доступ

Вывод четного числа в диапазоне 1-10

вывод простого числа в диапазоне 1-110

Задание

- 1) Написать функцию, которая принимает 2 числа и возвращает меньшее из них
- 2) Написать функцию, которая возводит переданное число в указанную степень
- 3) Создайте функцию определения знака числа $\text{sign}(x)$, которая вернет значение -1 , если аргумент « x » — отрицательное число, 1 — если положительное, 0 — если аргумент « x » равен нулю.
- 4) Написать функцию, которая принимает 2 числа и знак (+ - * /), считает пример и возвращает результат
- 5) Написать функцию, которая проверяет, является ли переданное ей число совершенным - натуральное число, равное сумме всех своих собственных делителей (то есть всех положительных делителей, отличных от самого числа) Например, число 6 равно сумме своих собственных делителей $6 = 1 + 2 + 3$. Пример совершенных чисел: 6, 28, 496
- 6) Написать функцию, которая принимает число и выводит таблицу умножения для этого числа