

Цель урока:

Вы поймете значимость функций в JavaScript, узнаете про их типы и сможете без проблем написать свою функцию, правильно назвать и использовать ее в коде.

Содержание урока:

1. Что такое функции в JavaScript?
2. Виды функций: function declaration, function expression, стрелочная функция.
3. Возврат значения из функции (через return и через стрелочную)
4. Как выбрать имя функции?

У стрелочных функций параметры передаются в круглые скобки.

Посмотрите на пример:

```
const showMessage = (message) => {  
  alert(`Ваше сообщение: ${message}`);  
}
```

У функции `showMessage` есть единственный параметр **message**. Когда у стрелочных функций есть только 1 параметр, то круглые скобки можно опустить.

```
const showMessage = message => {  
  alert(`Ваше сообщение: ${message}`);  
}
```

Но вот если параметров больше, чем 1, то круглые скобки обязательны, так как без них ничего работать не будет.

```
const showMessage = (message, personName) => {  
  alert(`${personName}, у вас новое сообщение!`);  
  alert(`Ваше сообщение: ${message}`);  
}
```

Задание #1

Как мы уже с вами разобрались, существует 3 вида функций: function expression, function declaration и стрелочная функция.

Первым делом потренируемся их создавать. Вам необходимо создать 3 функции трех разных видов. Они должны принимать в себя параметр name и возвращать строку,

подставляя туда этот параметр. Название у них должны быть следующими: **getName1**, **getName2**, **getName3**.

Функции необходимо удовлетворять следующим условиям:

1. В каждую из этих функций должен быть передан 1 параметр `name`.
2. Из каждой функции с помощью `return` должно быть возвращено значение "Имя равно `name`", где `name` - значение переданного в функцию параметра `name`.

После вам необходимо вызвать каждую функцию и вывести возвращаемое из нее значение в консоль.

Задание #2

Необходимо создать функцию **getSumOfNumbers**. Она будет считать сумму от 0 до переданного числа. Выберите любой из 3 типов при создании функции.

getSumOfNumbers принимает в себя 2 параметра: `number` и `type`.

Параметр `number` - это число, до которого будет считаться сумма. То есть, если было передано число 10, то функция должна посчитать сумму от 0 до 10 ($0 + 1 + 2 + \dots + 10$).

Параметр `type` отвечает за выбор чисел для подсчета суммы. Он может быть 3-мя значениями: "odd", "even" и "". Если `type` равняется "odd", то в сумму должны входить только нечетные числа, "even" - четные числа, пустая строка "" - все числа. По умолчанию параметр `type` должен быть равен `odd`.

Функция **getSumOfNumbers** должна возвращать итоговую сумму с помощью `return`.

Возможные результаты функции `getSumOfNumbers`:

1. `number = 10, type = 'odd'`. Возвращает 25.
 2. `number = 10, type = 'even'`. Возвращает 30.
 3. `number = 10, type = ''`. Возвращает 55.
-

Задание #3

Вам необходимо создать функцию **getDivisorsCount**, которая будет высчитывать количество делителей для переданного числа. Она принимает в себя единственный параметр **number**. По умолчанию он должен быть равен 1.

Если **number** меньше нуля либо является не целым (можно проверить при помощи функции **Number.isInteger(number)**), то выводите в модальное окно при помощи `alert` сообщение "number должен быть целым числом и больше нуля!".

Примеры:

```
getDivisorsCount(4) // Вернет 3 (делители - 1, 2, 4)
getDivisorsCount(5) // Вернет 2 (делители - 1, 5)
getDivisorsCount(12) // Вернет 6 (делители - 1, 2, 3, 4, 6, 12)
```

```
getDivisorsCount(30) // Вернет 8 (делители - 1, 2, 3, 5, 6, 10, 15, 30)
```

Задание #4 (дополнительное)

Помните, как в теме со строками, мы форматировали значения, которые вводит пользователь в текстовое поле? Так вот, представьте, что у нас есть 10 текстовых полей, для которых нам нужно сделать одно и то же форматирование. Это что же, придется писать 10 раз один и тот же код? Конечно же нет, ведь у нас существуют функции.

Часто функции нужны для того, чтобы реализовать какую либо логику (например, форматирование строки) и не копировать и вставлять ее несколько раз. Рассмотрим примеры ниже.

Без функции:

```
console.log('Привет, Максим!');
console.log('Вам 20 лет!');

console.log('Привет, Игорь!');
console.log('Вам 25 лет!');

console.log('Привет, Анастасия!');
console.log('Вам 28 лет!');
```

С функцией:

```
const getInfo = (name, age) => {
  console.log(`Привет, ${name}!`);
  console.log(`Вам ${age} лет!`);
}

getInfo('Максим', 20);
getInfo('Игорь', 25);
getInfo('Анастасия', 28);
```

Я думаю, вам понятно, что пример с функцией более удобен в использовании и уменьшает количество повторяющегося кода.

Сейчас ваша задача состоит в том, чтобы создать функцию **checkQuestionAnswer**, которая будет задавать пользователю вопрос и автоматически проверять полученный ответ.

Она принимает в себя 2 параметра: **question** и **correctAnswer**

Параметр question - это вопрос, который будет задаваться пользователю и будет передан, как параметр в функцию `prompt`.

Параметр correctAnswer - это правильный ответ на вопрос. Вам необходимо получить значение, которое введет пользователь в текстовое поле, и сверить его с параметром

correctAnswer. Если пользователь дал верный ответ, то выведите в модальном окне через alert сообщение “Ответ верный”, иначе - “Ответ неверный”.

Также ****в коде нам **нужно учесть**, если пользователь введет данную строку “яБлОко”, а параметр **correctAnswer** будет равен “Яблоко”, то ответ всё равно должен быть засчитан как правильный. То есть, ответ не должен зависеть от регистра символов и от количества пробелов в начале и в конце ответа.

Когда создадите функцию, вызовите данный код для ее проверки:

```
checkQuestionAnswer('Арбуз это фрукт или ягода?', 'Ягода');  
checkQuestionAnswer('Сколько в среднем зубов у взрослого человека?',  
'32');  
checkQuestionAnswer('Как называется самая маленькая птица в мире?',  
'Колибри');
```

Задание #5 (дополнительное)

До этого вы передавали в функцию такие типы данных как string, number, boolean. Но в JavaScript параметром функции можно передать все что угодно, даже функцию. Функция, которая передается как параметр в другую функцию, называется колбэк (callback). И вы сейчас с этим поработаете.

Для начала создадим 2 функции. Они будут нашими будущими колбеками:

1. showMessage - принимает в себя параметр message и выводит его в консоль через console.log
2. showErrorMessage - принимает в себя параметр message и выводит его в консоль через console.error (console.error тоже выводит сообщение в консоль, но красным цветом).

Далее необходимо создать еще одну функцию под названием checkTextOnErrorSymbol. С помощью нее мы будем искать в тексте запрещенный символ и выводить ошибку в консоль, если такой символ был найден.

checkTextOnErrorSymbol принимает в себя 4 параметра:

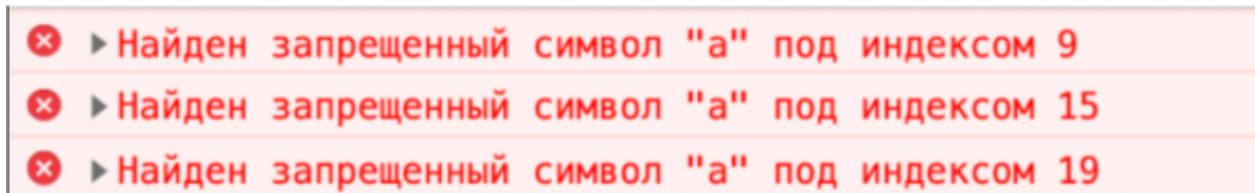
1. text, тип данных string. Отвечает за текст, в котором мы будем искать запрещенный символ
2. errorSymbol, тип данных string. Это запрещенный символ, который мы будем искать
3. successCallback - функция, которая будет выводить успешное сообщение, если запрещенных символов не было найдено
4. errorCallback - функция, которая будет выводить ошибку в консоль, если был найден запрещенный символ

Если запрещенный символ был найден, то вызовите колбэк errorCallback и передайте в него сообщение “Найден запрещенный символ “\${errorSymbol}” под индексом \${i}.” (замените errorSymbol на запрещенный символ, i - на индекс строки, в котором был обнаружен запрещенный символ).

Если ни одного запрещенного символа в строке не было найдено, то вызовите колбэк `successCallback` и передайте в него сообщение “В данном тексте нет запрещенных символов”. Проверяйте работоспособность вашей функции на данном коде:

```
const text = 'Привет! Как дела! Давно мы с тобой не виделись.';
checkTextOnErrorSymbol(text, 'a', showSuccessMessage,
showErrorMessage);
```

Для данного кода, в консоль должно быть выведено следующее:



```
✖ ▶ Найден запрещенный символ "a" под индексом 9
✖ ▶ Найден запрещенный символ "a" под индексом 15
✖ ▶ Найден запрещенный символ "a" под индексом 19
```

Подсказка:

Для того, чтобы получить символ строки по индексу, используйте следующую запись `text[index]`.