

Функции

Что такое функции?

- **функция** является, своего рода, строительным блоком, который решает элементарную задачу
 - Проименованная часть кода, которая может вызываться в других частях кода
 - Можно **передавать параметры** и **возвращать результат**
- Функции позволяют программистам строить программы "по кусочкам"

Почему надо использовать функции?

- Более **управляемое** программирование
 - **Разделение больших** проблем по **маленьким кусочкам**
 - **Улучшение организации** структуры программы
 - Улучшение **читабельности** и **восприятия** кода
 - введение **абстрактности**
- Повторное использование кода
 - Удобство **рефакторинга кода**

Объявление и создание функций

- функции могут именоваться
 - Используется для вызова функций
 - Описывает назначение функции
- В Функциях JavaScript не надо указывать возвращаемый тип данных
 - Каждая функция всегда возвращает значение

```
function printLogo() {  
  console.log("Основы JavaScript");  
  console.log("Курсы Веб-программирования");  
}
```

Способы определения функций

- Используя конструктор для объекта функции

```
var print = new Function("console.log('Hello')");
```

- С помощью объявления функции

```
function print() { console.log("Hello") };
```

- С помощью выражения функции

```
var print = function() { console.log("Hello") };  
var print = function printFunc() { console.log("Hello") };
```

Вызов функций

- Чтобы вызвать функцию достаточно использовать:
 - Имя функции
 - Круглые скобки
 - Точку с запятой (;)
 - Опционально но не обязательно
- Эти действия вызовут код функции на исполнение:

```
printLogo();
```

- Функция может вызывать:
 - Другую функцию
 - Саму себя (известна как **рекурсивная функция**)

```
function print(){  
  console.log("printed");  
}  
  
function anotherPrint(){  
  print();  
  anotherPrint();  
}
```

Параметры функции

- Для того, чтобы передать информацию в функцию необходимо указать **параметры** (которые называются **аргументами**)
 - Вы можете не передавать параметров, а можете передать их сколько угодно
 - Каждый параметр именуется
 - Параметры обычно являются значениями, которые используются внутри функции

- Параметры изменяют поведение функции в зависимости от того, какие значения они хранят

Установлене и использоание параметров функции

- Поведение функции зависит от переданных параметров
- Параметры могут быть любым типом
 - Number , String , Object , Array , И.Т.Д.
 - Даже быть Function (другой функцией)

```
function printSign(number) {  
  if (number > 0) {  
    console.log("Positive");  
  } else if (number < 0) {  
    console.log("Negative");  
  } else {  
    console.log("Zero");  
  }  
}
```

- Функции могут иметь столько параметров, сколько необходимо:

```
function printMax(x, y) {  
  var max;  
  x = +x; y = +y;  
  max = x;  
  
  if (y > max) {  
    max = y;  
  }  
  
  console.log(`Maximal number: ${max}`);  
}
```

Вызов функций с параметрами

- Для вызова функций и передачи параметров в качестве аргументов:
 - Используется имя функции и указание параметров перечисленных через запятую внутри круглых скобок

```
printSign(-5);  
printSign(balance);  
printSign(2 + 3);  
printMax(100, 200);  
printMax(oldQuantity * 1.5, quantity * 2);
```

- Вывести знак числа на экран

```
function printSign(number) {  
  number = +number;  
  
  if (number > 0) {  
    console.log(`Число ${number} положительное.`);  
  } else if (number < 0) {  
    console.log(`Число ${number} отрицательное.`);  
  } else {  
    console.log(`Число ${number} ноль.`);  
  }  
}
```

- вывести максимальное из 2х чисел

```
function printMax(x, y) {  
  var max = x;  
  
  if (max < y) {  
    max = y;  
  }  
  
  console.log(`Максимальное число: ${max}`);  
}
```

- создание функции, которая выводит треугольник из цифр:

n = 6	n = 5
1	
1 2	1
1 2 3	1 2
1 2 3 4	1 2 3
1 2 3 4 5	1 2 3 4
1 2 3 4 5 6	1 2 3 4 5
1 2 3 4 5	1 2 3 4
1 2 3 4	1 2 3
1 2 3	1 2
1 2	1
1	

```

function pringTriangle(n) {
    var line;
    n = +n;

    for (line = 1; line <= n; line += 1) {
        printLine(1, line);
    }

    for (line = n-1; line >= 1; line -= 1) {
        printLine(1, line);
    }
}

function printLine(start, end) {
    var line = "",
        i;
    start = +start; end = +end;
    for (i = start; i <= end; i += 1){
        line += " " + i;
    }
    console.log(line);
}

```

Объект arguments

- Каждая функция JavaScript имеет встроенный параметр `arguments`
 - он хранит информацию о параметрах передаваемых в функцию
 - Нет необходимости явного объявления
 - Содержится в любой функции

```

function printArguments() {
    var i;
    for(i in arguments) {
        console.log(arguments[i]);
    }
}
printArguments(1, 2, 3, 4); //1, 2, 3, 4

```

- объект `arguments` не является массивом
 - Он имеет похожую с массивами функциональность
- Если его надо перебрать, лучше его передать в массив:

```
function printArguments() {
    var i,
        args;

    args = [].slice.apply(arguments);
    for(i in args) {
        console.log(args[i]);
    }
}

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

Возврат значений из функции

- Каждая функция в JavaScript возвращает значения
 - Возвращает `undefined` если не указывается возвращаемое значение
 - Может быть любым типом
 - `Number`, `String`, `Object`, `Function`

```
var head = arr.shift();
var price = getPrice() * quantity * 1.20;
var noValue = arr.sort();
```

- Функции могут возвращать любой тип данных:
 - `Number`, `String`, `Object`, и.т.д.
- Используйте `return` ключевое слово для возвращения результата

```
function multiply (firstNum, secondNum) {
    return firstNum * secondNum;
}

function sum (numbers) {
    var sum = 0, number;
    for(number of numbers){
        sum += number;
    }
    return sum;
}
```

`return` утверждение

- `return` утверждение:
 - Немедленно прерывает выполнение функции
 - Возвращает указанное значение
- Для того, чтобы прервать выполнение функции просто вызовите:

```
return;
```

- Return может быть указано сколько угодно раз внутри функции
 - Для того, чтобы вернуть определенное значение в определенной ветке функции
- Проверка числа на простоту:

```
function isPrime(number){
    var divider,
        maxDivider;

    number = +number;
    maxDivider = Math.sqrt(number);

    for(divider = 2; divider <= maxDivider; divider += 1){
        if(number % divider === 0) {
            return false;
        }
    }
    return true;
}
```

- Вычислить сумму всех содержащихся в массиве четных чисел

```
function sum(numbers) {
    var number,
        sum = 0;

    for (number of numbers) {
        if (0 === number % 2) {
            sum += number;
        }
    }
    return sum;
}
```

Области видимости функций

- Каждая переменная может использоваться в определенной для нее области видимости
 - Область определяет то, где переменная может использоваться
 - Обычно применяется **локальная** и **глобальная** область видимости

```
var arr = [1, 2, 3, 4, 5, 6, 7];

function countOccurrences (value) {
  var item,
    count = 0;
  for (item of arr) {
    if (item === value) {
      count++;
    }
  }

  return count;
}
```

- `arr` определена в глобальной области видимости (это значит может использоваться везде)
- `count` объявлена внутри `countOccurrences` и может использоваться только внутри нее
- Попробуйте убрать `var` перед `count`

Перегрузка функций

- JavaScript не поддерживает перегрузку функций
 - т.е функции с одинаковым именем перекрывают друг друга

```
function print(number) {
  console.log(`Number: ${number}`);
}

function print(number, text) {
  console.log(`Number: ${number}\nText: ${text}`);
}

print(2);
//prints:
//Number: 2
//Text: undefined`
```

Перегрузка функций в JavaScript

- Перегрузку функций в JavaScript можно подделать
 - т.е сделать похожей на перегрузку
 - Существует несколько способов как это сделать
- Разное число параметров:


```
function printText (number, text) {
  switch (arguments.length) {
    case 1 : console.log (`Number: ${number}`);
      break;
    case 2 :
      console.log (`Number: ${number}`);
      console.log (`Text: ${text}`);
      break;
  }
}

printText (5); //logs 5
printText (5, "Lorem Ipsum"); //logs 5 and Lorem Ipsum
```

- Разные типы параметров:

```
function printValue (value) {
  switch (typeof value) {
    case "number" : console.log (`Number: ${value}`); break;
    case "string" : console.log (`String: ${value}`); break;
    case "object" : console.log (`Object: ${value}`); break;
    case "boolean" : console.log (`Number: ${value}`); break;
  }
}

printValue (5);
printValue ("Lorem Ipsum");
printValue ([1, 2, 3, 4]);
printValue (true);
```

Перегрузка с помощью параметров по умолчанию

- в JavaScript все параметры опциональны
 - функция может быть вызвана без их указания
- Параметры по умолчанию проверяются в теле функции
 - Если параметр не указан, присвоить ему значение

```
function getRandomValue(str, start, end){
  start = start || 0;
  end = end || str.length;
  //function code
}
```

- Для создания функции с параметром опции
 - Создайте функцию с одним параметром
 - Каждый параметр является параметром передаваемого объекта

```
function getRandomValue(opt) {  
  var min = +opt.min || Number.MIN_VALUE;  
  var max = +opt.max || Number.MAX_VALUE;  
  
  return (Math.random() * (max - min + 1) + min) | 0;  
}  
  
console.log(getRandomValue({min:0, max: 15}));
```