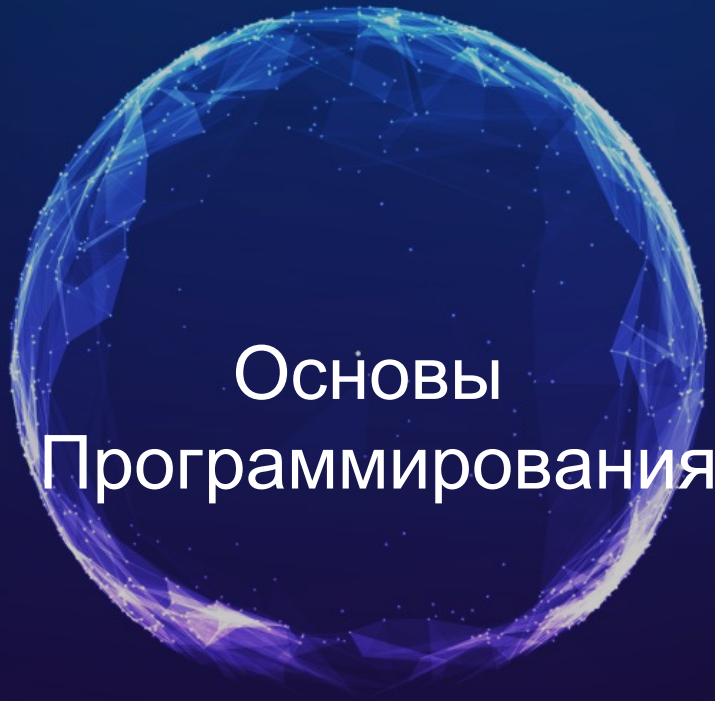

Основы Программирования

Занятие №6
Функции

IT Education Academy

WWW.ITEA.UA

План занятия

- Функция
- Переменные в функции
- Стек
- Рекурсия

Повторение материала

Функция

Функция - это подпрограмма, к которой можно обратиться из другого участка кода. Обычно, функция обладает именем, с помощью которого к ней можно обратиться.

Состоит из тела (блока команд), может принимать аргументы и возвращать результат выполнения.

Функции нужны для избавления от повторяющихся участков кода, а также логического разделения. Например, функция сортировки массива.

Функция в JavaScript

Синтаксис создания функции:

```
function name(arg1, arg2, arg3) {
  //тело функции (набор команд)
}
```

Команды, описанные в теле функции будут выполнены только когда функция будет вызвана.

Аргументы функции выступают как обычные переменные, значения которых можно считывать и перезаписывать.

Функция в JavaScript

Для вызова функции используется синтаксис: name(arg1, arg2, arg3):

```
function sayHi(name) {
  console.log(‘hi ’ + name);
}

sayHi(‘Vlad’);
sayHi(‘Andrew’);
sayHi(‘John’);
```

Функция в JavaScript

Написать функцию, которая принимает два аргумента имя и возраст и выводит на экран «Привет __, через 5 лет тебе будет __ лет.». Назвать ее processUser. Вызвать функцию 3 раза с разными аргументами.

Важность функций

Большим преимуществом функций является то, что логика работы находится в одном месте. И поменяв код тела функции, автоматически во всех местах где вызывается эта функция будут доступны изменения.

Если понадобится поменять сообщение – достаточно изменить его в одном месте: в функции, которая его выводит.

Важность функций

Изменить логику функции processUser следующим образом: проверять совершеннолетний ли пользователь, если да, выводить «Добро пожаловать», иначе «Приходите через __ лет».

Переменные

В теле функции можно обьявлять переменные.

```
var global = '...';
function example(local1) {
  var local2 = local1 * 1;
  console.log("This is number: " + local2);
}
example(5);
console.log(global);
console.log(local1); // Ошибка!
console.log(local2); // Ошибка!
```

Такие переменные называются локальными (они существуют только внутри функции).

Переменные, которые мы обьявляли до этого - глобальные (они существуют до завершения программы). Внутри функций можно использовать глобальные переменные.

Глобальные переменные

```
var password = ‘qwerty’;
function testPassword(pwd) {
  if (pwd === password)
    console.log(‘correct’);
  else
    console.log(‘wrong password’);
}
testPassword(‘123456’);
testPassword(‘qwerty’);
```

Глобальные переменные

Если внутри функции создать переменную с таким же именем как и глобальная, внутри функции будет видно значение локальной переменной.

```
var number = 10;
function test (n) {
  var number = n + 10;
  console.log(number);
}
console.log(number); //10
test(50); //60
console.log(number); //10
```

Участок кода, где можно использовать ту, или иную переменную называется областью видимости.

Глобальные переменные

Написать функцию globalExample(), которая выводит на экран “Вы меня вызывали _ раз(а)”. И каждый вызов увеличивает это значение на 1. Например:

globalExample(); //Вы меня вызвали 1 раз(а)
globalExample(); //Вы меня вызвали 2 раз(а)
globalExample(); //Вы меня вызвали 3 раз(а)
globalExample(); //Вы меня вызвали 4 раз(а)
globalExample(); //Вы меня вызвали 5 раз(а)

Необязательные аргументы

В JavaScript необязательно передавать все аргументы в функцию.
Например:

```
function example(a, b, c) {…}
example(1, 2, 3);
example(1, 2);
example(1);
example();
```

Тем переменным, в которые не были переданы аргументы присваивается значение `undefined`.

```
function example2(required, optional) {
  if (optional === undefined)
    optional = 1; //установить значение по умолчанию
  ...
}
```

Необязательные аргументы

Написать функцию argumentsExample которая принимает три аргумента и выводит на экран значение каждого из них, если какой-то из аргументов передан не был, присвоить ему значение по умолчанию “empty”.

Возврат результата

Функции, которые просто выводят на экран результат, не очень полезные. Поэтому есть возможность возвращать результат выполнения функции, чтобы его можно было использовать в месте вызова.

Например, если бы существовала функция sum(a, b) которая возвращает результат сложения двух чисел, то этот результат можно было бы присвоить переменной:

```
function sum(a, b) {…}

var result1 = sum(1, 2); // result1 === 3
var result2 = sum(sum(1, 2), 3); // result2 === 6

var result3 = sum(result1, result2); // result3 === 9
```

Возврат результата

В JavaScript для возврата результата используется команда `return`.

```
function sum(a, b) {
  return a + b;
}

var result1 = sum(1, 2); // result1 === 3
var result2 = sum(sum(1, 2), 3); // result2 === 6

var result3 = sum(result1, result2); // result3 === 9
```

Возврат результата

Если явно не указать `return`, интерпретатор в конце тела функции «подставит» `return undefined`.

```
function show1(a) {
  console.log('hi' + a);
}
function show2(a) {
  console.log('hi' + a);
  return undefined;
}
```

`console.log()` - также является функцией (однако ее тело написано не на JavaScript), результатом ее выполнения является `undefined`.
Node.js после выполнения каждой команды выводит на экран результат выполнения. Например, написав Math.pow(2,3), вы увидите 8. Именно поэтому, каждый раз используя console.log(…) вы видели undefined - результат выполнения этой функции.

Функции

1. Написать функцию НОД.
2. Написать функцию наименьшего общего кратного, которое использует НОД: НОК(a, b) = (a * b) / НОД(a, b).
3. Написать функцию поиска максимального числа в массиве.

Стек

Это структура данных, которая хранит информацию, для функции.
В стеке находятся:

- место возврата (место откуда эта функция была вызвана)
- аргументы функции
- локальные переменные.

Стек находится в оперативной памяти. Каждый вызов функции добавляет запись в стек, каждый возврат удаляет.

Стек

```
function f(a) {
  return a + 5;
}
function g(a) {
  return f(a) * 2;
}
function h(a) {
  return g(a) - 10;
}
function j(a) {
  return h(a) / 4;
}

console.log(j(100));
```

console.log(50)
h(100) // 200
g(100) //210
f(100) // 105

Рекурсия

Рекурсия - это вызов функции из нее же самой.
При рекурсивном вызове важно определить условие окончания рекурсии. А также понимать тот факт, что каждом вызове создается новая запись в стеке и он может быть переполненным.

```
function pow(x, y) {
  if (y == 0) return 1;
  else return x * pow(x, y -1);
}

pow(2, 4); // сколько записей в стеке будет создано?
```

Рекурсия

Написать функцию вычисления факториала, используя рекурсию.

n! = n * (n - 1) * (n - 2) * … * 3 * 2 * 1
или
n! = n * (n - 1)!, а 1! = 1

Summary

1. Функция - это подпрограмма с определенным назначением.
2. Для возврата значения - return.
3. Аргументы - это переменные, в которые записываются параметры функции.
4. В стеке хранится информация для функции.
5. Рекурсия - вызов функции из себя самой.
6. Локальные переменные видны только внутри функции.

Домашнее задание

1. Написать функцию возведения в степень: pow(x, y), где y - необязательный аргумент и значение по умолчанию - 2.
2. Написать функцию show(a), которая выводит на экран матрицу чисел (массив массивов) [[0,1, 2], [3, 4, 5], [6, 7, 8]]:

0 1 2
3 4 5
6 7 8

3. Написать функцию нахождения числа фибоначчи fib(n), которое определяется следующим образом:
fib(0) = 1,
fib(1) = 1
fin(n) = fin(n - 2) + fin(n - 1)
Первые 10 чисел: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
Найти значение для n = 10, 15, 20, 25, 30, 35, 40, 45. Примерно определить сколько времени работала программа для каждого из случаев (субьективно «на глаз»).
** Написать функцию нахождения индекса минимального числа в массиве начиная с определенной позиции minIdx(array, startIndex). Написать функцию сортировки массива методом выбора, используя функцию minIdx.

Спасибо за внимание!