

# Функции в JavaScript

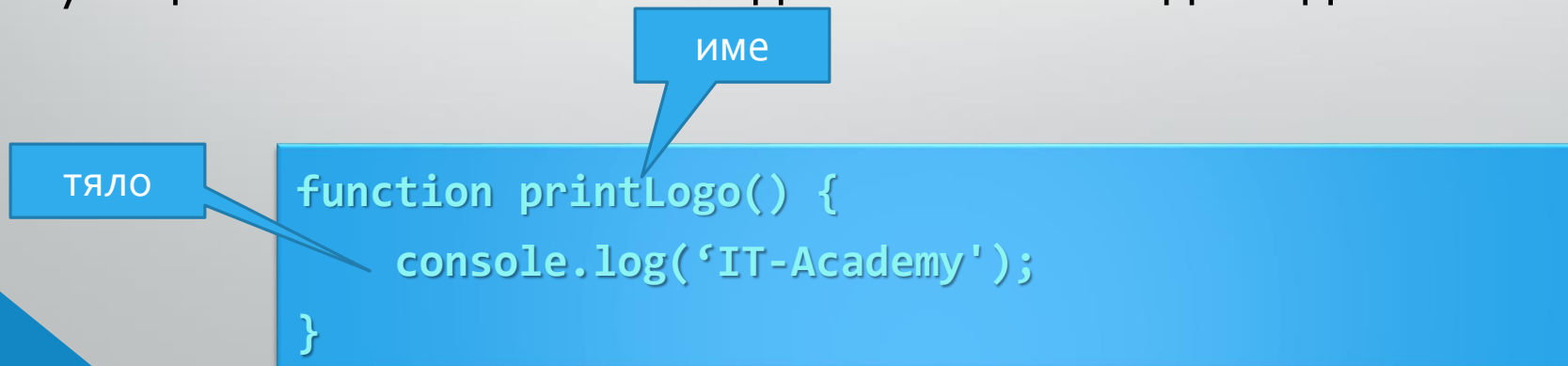
Димитър Митев

# Защо да използваме функции?

- Разделят кода на малки, преизползваеми парчета код
- Съдържат конкретна логика
- Повишават нивото на абстракция
- Подобряват четимостта на кода
- По-лесна поддръжка на кода в бъдеще

# Функции (functions)

- Функции се използват за повтаряеми парчета код
- Функциите представляват малки парчета код, които си имат имена и могат да връщат стойност
- Функциите могат да бъдат извиквани от други функции или в други функции; Функцията може да се извиква и сама, в своето тяло ([рекурсия](#))
- Функцията има тяло и може да има име или да бъде анонимна



```
function printLogo() {  
  console.log('IT-Academy');  
}
```

Diagram illustrating a function definition:

- The label **име** (name) points to the function name `printLogo()`.
- The label **тяло** (body) points to the function body, which contains the code `console.log('IT-Academy');`.

# Деклариране на функции

- Чрез конструктора на *Function* обекта
  - `var sayHello = new Function('console.log("Hello")');`
- Функция декларация (*function declaration*)
  - `function sayHello() { console.log('Hello') };`
- Функция израз (*function expression*)
  - `var sayHello = function() { console.log('Hello') };`
- Препоръчва се името на функцията да бъде описателно и да започва с глагол

# Функции с параметри

- На функциите могат да се подават параметри (аргументи)
  - Могат да се подадат 0 или повече параметъра
  - Всеки параметър си има име, което е еднозначно определено за функцията
  - Параметрите приемат определените им стойности едва когато функцията е извикана
- Параметрите могат да променят поведението на функцията според техните стойности
- Параметрите могат да бъдат от всякакъв тип, дори и друга функция

# Извикване на функции

- Функцията се извиква чрез
  - Името на функцията
  - Скоби (), в () с изреждат параметрите
  - ;
- Когато функцията бива извикана, нейното тяло се изпълнява и се връща съответния резултат
- Функциите в JS нямат тип на върната стойност както в други езици

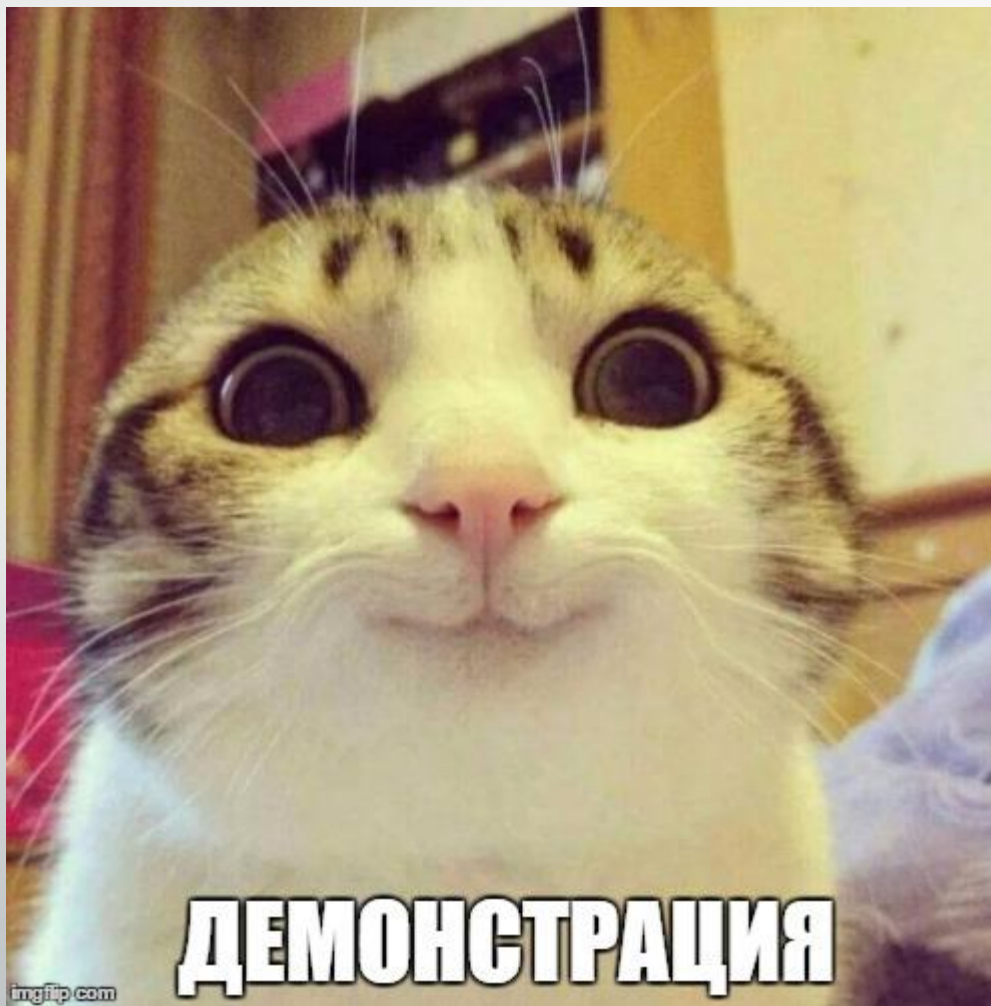
```
function printLogo() {  
    console.log('IT-Academy');  
}  
  
printLogo();
```

## *arguments* обекта

- Всяка функция има специален обект наречен *arguments*
  - няма нужда да бъде предварително деклариран
  - Съдържа информация за параметрите на функцията
  - Всяка функция го има, без значение дали има подадени параметри или не
- Обектът *arguments* доста прилича на масив, но всъщност не е
  - Ако ще го използвате като масив, то трябва да бъде преобразуван към масив

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

# Извикване и деклариране на функции

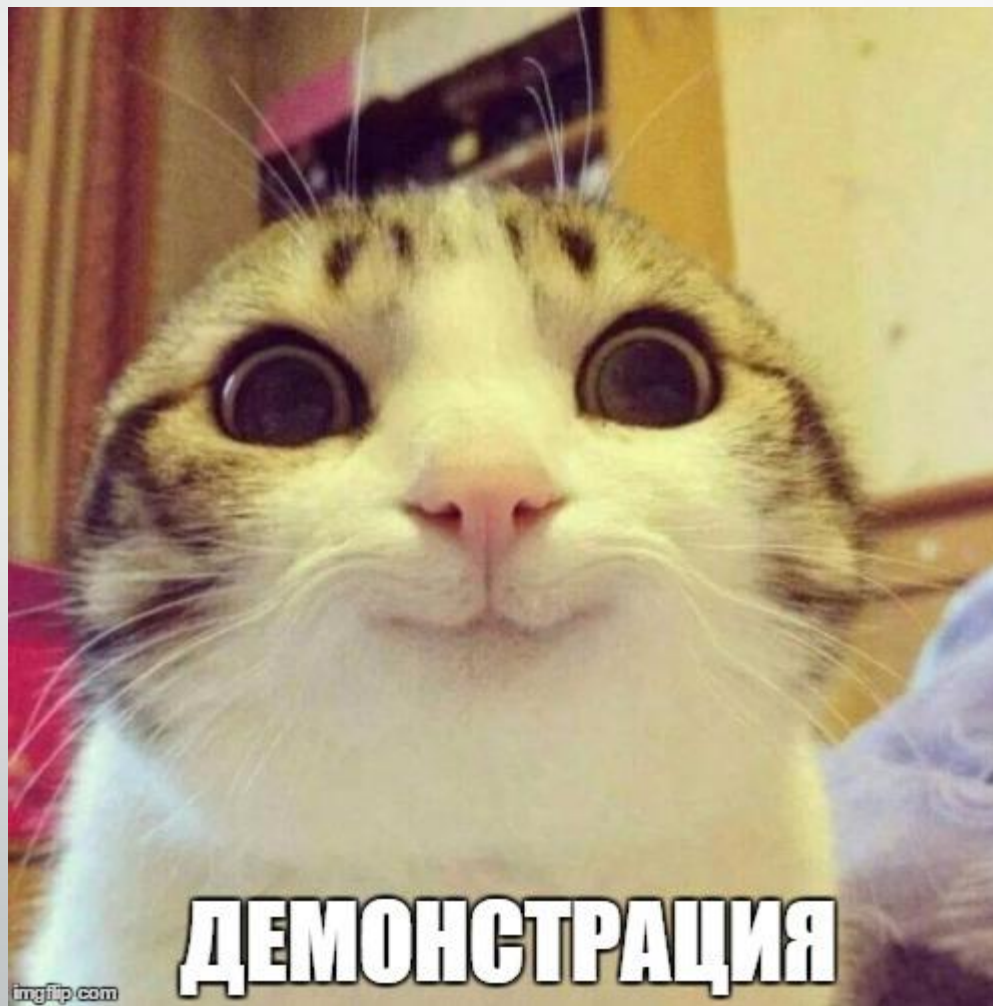




# Функции, които връщат стойност

- По дизайн всяка функция в JS връща стойност
  - Ако не бъде изришно върната стойност от програмиста, то функцията връща *undefined*
  - Връщаната стойност може да бъде от всякакъв тип – *Number, String, Object, Function*
- За да бъде върната стойност се използва кл. дума *return*
  - *return* връща директно стойността от изпълнението на функцията
  - *return* прекратява изпълнението на функцията моментално
  - В 1 функцията може да има повече и 1 *return* според логиката, която съдържа, но ще се изпълни само 1
  - Винаги използвайте *return* с ; (точка и запетая)

Функции, които връщат стойност



# Overloading на функции

- В JS не се поддържа overloading на функциите
- Последното въведено презаписва горните
- Overloading може да бъде „фалшифициран“ чрез *arguments* обекта

```
function print(number) {  
    console.log('Number: ' + number);  
}  
  
function print(number, text) {  
    console.log('Number: ' + number +  
        '\nText: ' + text);  
}  
  
print(2);
```

# „Фалшифициране“ на overloading на функции

- Използването на *switch-case* конструкцията не се счита за добра практика в програмирането

```
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
```

# Параметри по подразбиране

- *Параметри по подразбиране за пръв път са представени в EcmaScript2015*
  - *Поддържат се само в последните версии на браузърите*
  - *Необходими са shim-ове за по-стари версии*
- Все пак JS дава възможност да бъдат зададени дефолтни стойности, ако няма такива въведени

```
function calculateSurface(a = 5, b = 8){  
    return a * b;  
}
```

```
function calculateSurface(a, b) {  
    a = a || 5;  
    b = b || 8;  
  
    return a * b;  
}
```

# Обхват на функциите

Променлива, достъпна  
от целия код

- Обхвата на функциите дефинира къде техните променливи са променливи
- В JS света реално съществуват **локален** и **глобален** обхват (*local and global scope*)
- Тялото на функцията е единственото нещо в JS, което има собствен обхват
- Всички обекти, извън функция, са в глобалния scope
  - Добре е да се използва "use strict"

```
var arr = [1, 2, 3, 4, 5, 6, 7];  
function countOccurrences (value) {  
  var count = 0;  
  for (var i=0; i < arr.length; i++) {  
    if (arr[i] === value) {  
      count++;  
    }  
  }  
  return count;  
}
```

Променлива достъпна само в  
тялото на функцията

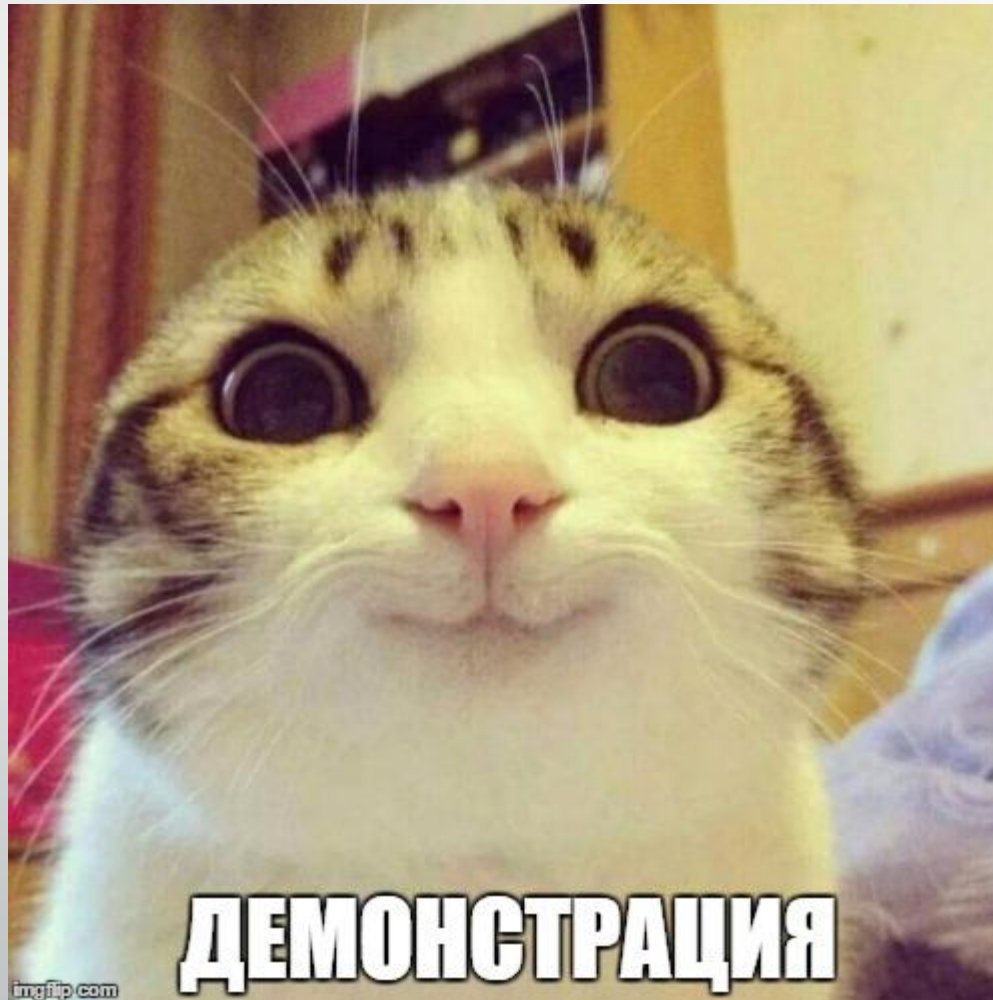
# Обхват на функциите(1)

- Референциите (връзките) към обектите в JS спазват правилото, че винаги се отнасят за най-близкото
- Ако имаме функцията, която дефинира обект и в нея имаме друга функцията, която дефинира обект със същото име, то всяка функция има собствен обхват и обекта е дефиниран само в конкретната функция
- Може да се използва ключовата дума *let*
  - Идва с **ES6**
  - Може да се използва само на последните версии браузъри
  - Създава т.нар. „блоков обхват“ (block scope), напр. в рамките на *if*-конструкция
  - [Why was block scope not originally implemented in JS?](#)

```
function outer() {  
    var x = 'OUTER';  
  
    function inner() {  
        var x = 'INNER';  
        return x;  
    }  
  
    return {  
        x: x,  
        f: inner()  
    };  
}  
  
console.log(outer());
```



# Обхват на фунцкиите





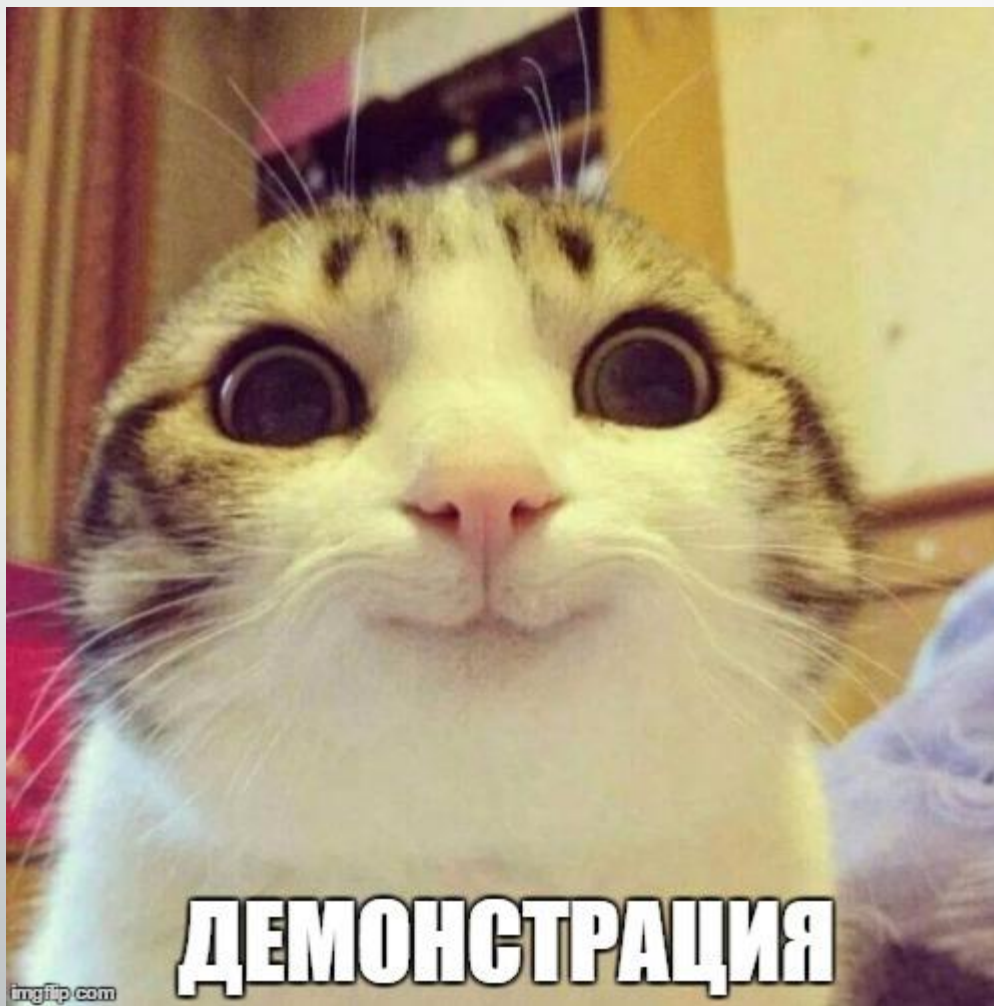
# Приближения (closures)

- Приближенията (closures) са специален вид структура в JS
- Комбинират функцията, както и нейния контекст
- Чрез приближенията (closures) можем да скрием обекти във дадена функция от външния свят (енкапсулация)

```
function outer(x){  
  function inner(y){  
    return x + " " + y;  
  }  
  return inner;  
}
```

**inner()** оформя приближение.  
Съдържа референция към **x**

# Приближения (closures)

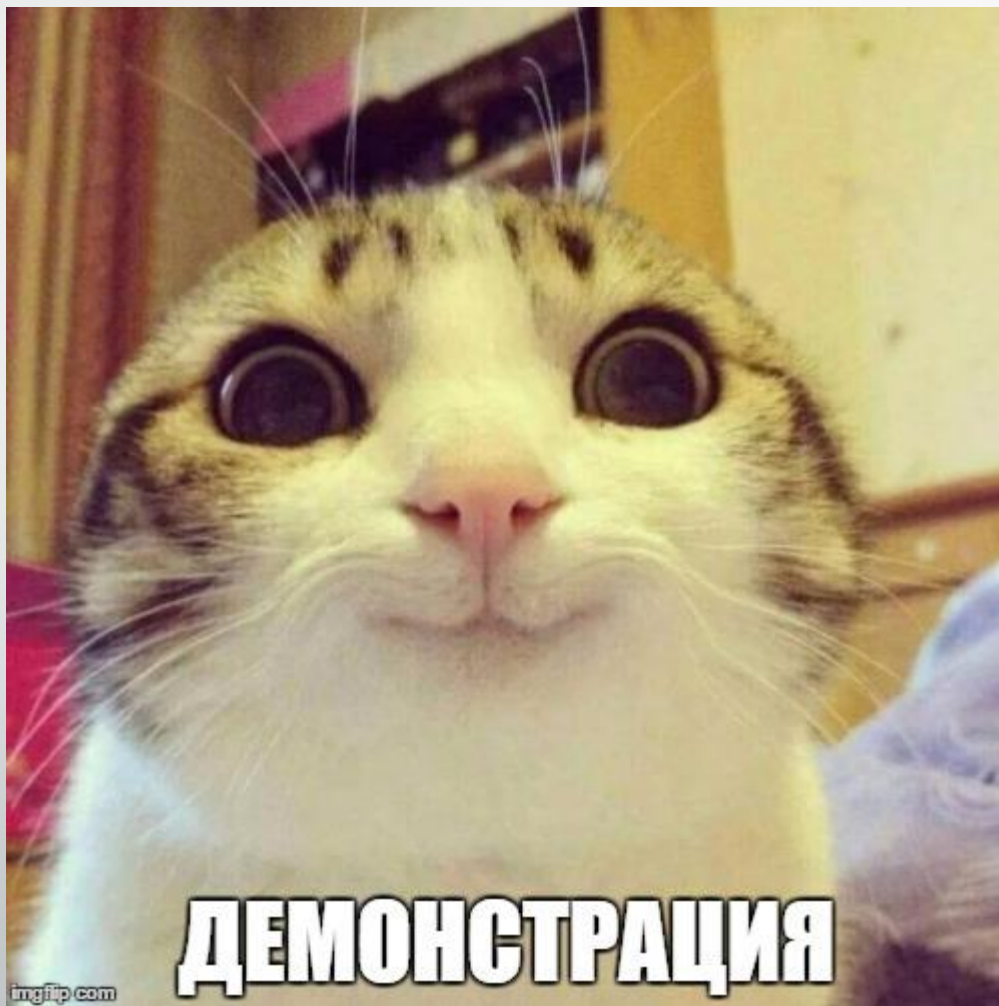


# Immediately invoked function expressions

- IIFE -> immediately invoked function expression
- Функцията се изпълнява в момента на своето деклариране
- Използват се за създаване на *scope*, т.е. да ограничим видимостта на дадени променливи/функции
- Може да бъде дефиниран стриктен режим на работа, т.е. не се позволява да има деклариране на променливи без *var/let*

```
(function() {  
    "use strict";  
  
    console.log("Hello from the IIFE!");  
})();
```

IFE

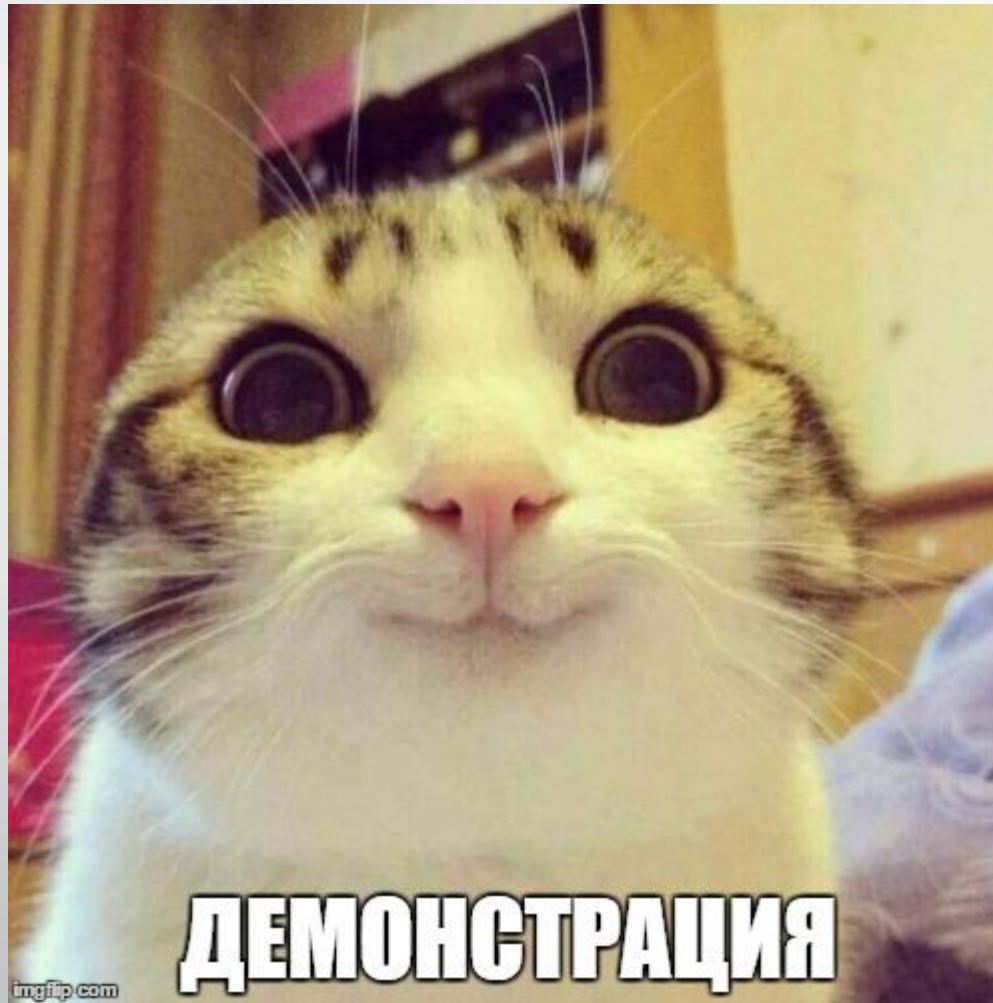


# Рекурсия

- Всяка функция може да бъде извикана отново в своето тяло
- Рекурсията винаги трябва да има „дъно“, т.е. условие, при което се прекратява извикването на функцията в себе си
- Рекурсията е много удобен способ за обхождане на неизвестни колекции
- Всяка рекурсия може да бъде заменена чрез итеративно решение
  - В някои случаи рекурсията е много по-удобна от итерацията

```
var fact = function (n) {  
  if (n === 0) {  
    return 1;  
  }  
  
  return n * fact (n - 1);  
};  
  
console.log(fact(5)); // 120
```

# Рекурсия





**НЯКОЙ ИМА ЛИ**



**ВЪПРОСИ?**

# Домашна работа

1. Напишете JS функция, която да връща последната цифра в дадено число като дума на английски. Напр. 305 -> five
2. Напишете JS функция, която да намира колко пъти се среща дадена дума в даден текст. Търсеното може да бъде чувствително към главни букви или да не бъде чувствително, нека това зависи от параметър, който да има дефолтна стойност.
3. Напишете JS функция, която да намира 1вия елемент от даден масив от цели числа, който е по-голям от неговите съседни и да връща индекса му или -1, ако такъв няма. Напр. [1,3,2,5,6] -> 1
4. Напишете JS функция, която да калкулира дадени числа. Възможните операции са събиране, изваждане, умножение, деление, деление с остатък.
5. Напишете JS функция, която получава текст и връща нов текст, в който всяко изречение е на нов ред.