

# 05. ФУНКЦИИ И ТО, ЧТО С НИМИ СВЯЗАНО

*курс лекций по информатике и программированию  
для студентов первого курса ИТИС КФУ (java-поток)  
2023/2024*

**М.М. Абрамский**

кандидат технических наук, доцент кафедры программной инженерии

**СНАЧАЛА ДОГОНИМ ТО, ЧТО  
НЕ УСПЕЛИ В ТОТ РАЗ**

# Полезные штуки для массивов

Инициализация в коде.

```
int[] arr = new int[] {4, 8, 15, 16, 23, 42};
```

```
int[] arr = {4, 8, 15, 16, 23, 42};
```

Зачем?

Как узнать размер?

# Многомерные массивы

- Пример:

```
int n = 10;  
int m = 20;  
int [][] arr = new int[m][n];
```

- Обращение к элементу:  $a[i][j]$
- Если  $a$  – двумерный массив, то  $a[i]$  – это что?

# Есть только одномерные массивы

```
int n = 10;  
int m = 20;  
int [][] arr = new int[m][n]; // T[], где T - это массив
```

- arr – одномерные массив из элементов типа «одномерный массив»
- arr[i] – это элемент массива (в нашем случае – одномерный массив)

# Загадка

*какова сложность поиска максимума для  
матрицы  $n \times n$ ?*

# Ступенчатые массивы

Не всегда все подмассивы в многомерном массиве должны быть одного размера

```
int n = 10;  
int [][] a = new int[n] [];  
a[0] = new int[n];  
a[1] = new int[n-1];  
//...
```

# TOP DOWN DESIGN



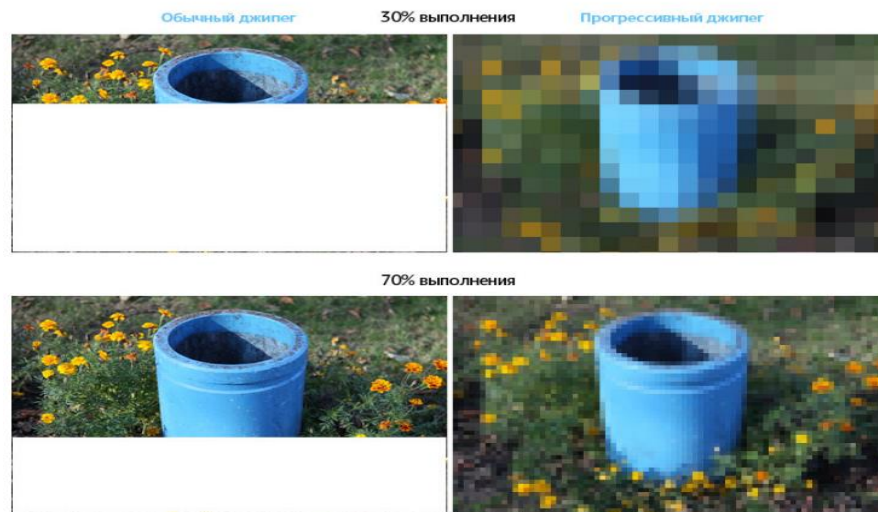
# «Метод прогрессивного джипега»

Метод прогрессивного  
jpeg-a

<https://www.artlebedev.ru/kovodstvo/sections/167/>

«В любую секунду любой проект готов на 100%, хотя проработанность может быть и на 4%»

Метод прогрессивного джипега  
составлен Артемием Лебедевым



# Top-Down проектирование (здесь)

- Программа всегда написана. Мы лишь конкретизируем ее части.
- Задача: вывести треугольник из единиц, высота треугольника вводится ( $n$ ). Пример для  $n = 5$ :

```
1
111
11111
1111111
111111111
```

# «Программа уже написана»

```
public static void main(String[] args) {  
  
    // TODO ввод n;  
    // TODO вывод треугольника с высотой n;  
  
}
```

# Ввод данных понятен

```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    // TODO вывод треугольника с высотой n;  
  
}
```

# Подумаем

Для  $n = 5$ :

1	1
2	111
3	11111
4	1111111
5	111111111

Количество строк = высота =  $n$

```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    // TODO вывод треугольника с высотой n;  
  
}
```



```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    // TODO вывод n строк;  
  
}
```

```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    // TODO вывод треугольника с высотой n;  
  
}
```



```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    for (int i = 1; i <= n; i++) {  
        // TODO вывод строки с номером i;  
    }  
  
}
```

# Подумаем еще

- Каждая строка:
  - сначала некоторое количество пробелов,
  - затем некоторое количество единиц,
  - затем перенос строки

1	1
2	111
3	11111
4	1111111
5	111111111



```
public static void main(String[] args) {

    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод строки с номером i;
    }

}
```



```
public static void main(String[] args) {

    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод пробелов;
        //TODO вывод единиц;
        //TODO вывод переноса строки;
    }

}
```

```
public static void main(String[] args) {

    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод строки с номером i;
    }

}
```



```
public static void main(String[] args) {

    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод пробелов;
        //TODO вывод единиц;
        System.out.println();
    }

}
```

# И еще подумаем

- Оценим количество единиц для каждой строки:
  - 1 строка – 1 единица
  - 2 строка – 3
  - 3 строка – 5
  - ...
  - $i$ -я строка –  $(2i - 1)$**

1	1
2	1 1 1
3	1 1 1 1 1
4	1 1 1 1 1 1 1
5	1 1 1 1 1 1 1 1 1

# Для пробелов

- Последняя строка (под номером  $n$ ) пробелов не содержит.
- Предпоследняя строка ( $n - 1$ ) содержит 1 пробел
- Строка перед ней ( $n - 2$ ) содержит 2 пробела

...

- В  $i$ -й строке:  $n - i$

1	0 0 0 0 1
2	0 0 0 1 1 1
3	0 0 1 1 1 1 1
4	0 1 1 1 1 1 1 1
5	1 1 1 1 1 1 1 1 1

```
public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод пробелов;
        //TODO вывод единиц;
        System.out.println();
    }
}
```



```
public static void main(String[] args) {

    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод n - i пробелов;
        //TODO вывод 2i - 1 единиц;
        System.out.println();
    }

}
```

```

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        //TODO вывод n - i пробелов;
        //TODO вывод 2i - 1 единиц;
        System.out.println();
    }
}
    
```



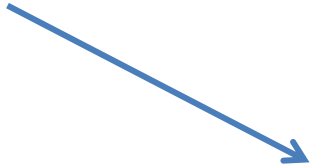
```

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) {
            System.out.print(' ');
        }
        for (int j = 1; j <= 2 * i - 1; j++) {
            System.out.print('1');
        }
        System.out.println();
    }
}
    
```

# Спроектировано Top-Down

```

public static void main(String[] args) {
    // TODO ввод n;
    // TODO вывод треугольника с высотой n
}
    
```



```

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) {
            System.out.print(' ');
        }
        for (int j = 1; j <= 2 * i - 1; j++) {
            System.out.print('1');
        }
        System.out.println();
    }
}
    
```

# Что-то в коде должно смутить

```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    for (int i = 1; i <= n; i++) {  
        for (int j = 1; j <= n - i; j++) {  
            System.out.print(' ');  
        }  
        for (int j = 1; j <= 2 * i - 1; j++) {  
            System.out.print('1');  
        }  
        System.out.println();  
    }  
}
```



# Для наглядности

```
for (int i = 0; i < n; i++) {  
    System.out.println('0');  
}  
for (int i = 0; i < 100; i++) {  
    System.out.println('a');  
}  
for (int i = 0; i < 200; i++) {  
    System.out.println('1');  
}  
for (int i = 0; i < n / 2; i++) {  
    System.out.println('%');  
}
```

# Функции! Цели использования

- Устранение дублирование кода
- Структуризация, реализация TOP-DOWN подхода
  - Помните, писали //TODO?
  - А теперь можно сразу метод/функцию писать.
- Удобство чтения («инкапсуляция»)
  - Если написан вызов факториала, то зачем мне лезть в его реализацию?

# Вынужденные ограничения этой лекции

- Все функции объявляем рядом с `main`, в том же классе, где и `main`
- У всех функций в Java ставим **`public static`**
- В рамках данной лекции «метод» и «функция» – *синонимы*;

# Объявление функций

```
public static int factorial(int n) {  
    int p = 1;  
    for (int i = 1; i <= n; i++) {  
        p *= i;  
    }  
    return p;  
}
```

```
public static boolean arrayHasZero(int [] array) {  
    for (int i = 0; i < array.length; i++) {  
        if (array[i] == 0)  
            return true;  
    }  
    return false;  
}
```

# Локальные переменные

```
public static int factorial(int n) {
    int p = 1;
    for (int i = 1; i <= n; i++) {
        p *= i;
    }
    return p;
}
```

Ничего общего у обоих `i` нет

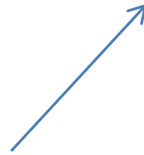
```
public static boolean arrayHasZero(int[] array) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == 0)
            return true;
    }
    return false;
}
```

Локальные переменные  
существуют лишь в методе

`main` – тоже метод,  
его переменные - локальные

# Вызов метода. Смысл return

```
public static void main(String[] args) {  
    int [] a = new int[20];  
    // Ввод массива был  
    System.out.println(arrayHasZero(a));  
}
```



main приостановился  
управление передалось в метод arrayHasZero  
когда он выполнится, оно **вернется** в это место  
с чем вернется?  
main продолжит работу

# Решение для треугольника – без функций

```
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    for (int i = 1; i <= n; i++) {  
        for (int j = 1; j <= n - i; j++) {  
            System.out.print(' ');  
        }  
        for (int j = 1; j <= 2 * i - 1; j++) {  
            System.out.print('1');  
        }  
        System.out.println();  
    }  
}
```

# Решение для треугольника — с функциями

```
public static void printChar(char c, int n) {  
    for (int j = 1; j <= n; j++) {  
        System.out.print(c);  
    }  
}  
  
public static void main(String[] args) {  
  
    int n = Integer.parseInt(args[0]);  
    for (int i = 1; i <= n; i++) {  
        printChar(' ', n - i);  
        printChar('1', 2 * i - 1);  
        System.out.println();  
    }  
}
```



# Решение для треугольника — с функциями [2]

```
public static void printChar(char c, int count) {  
    for (int j = 1; j <= count; j++) {  
        System.out.print(c);  
    }  
}  
  
public static void printTriangle(int n) {  
    for (int i = 1; i <= n; i++) {  
        printChar(' ', n - i);  
        printChar('1', 2 * i - 1);  
        System.out.println();  
    }  
}  
  
public static void main(String[] args) {  
    int n = Integer.parseInt(args[0]);  
    printTriangle(n);  
}
```

# «Вас вызывают!»

```

public class MyClass {

    public static void f() {
        System.out.println("f");
    }
    public static void g() {
        f();
        System.out.println("g");
    }
    public static void h() {
        g();
        System.out.println("h");
    }
    public static void main(...) {
        h();
        System.out.println("main");
    }
}
    
```

## Трасса:

- main вызвал h и ждет его конца
- h вызвал g и ждет его конца
- g вызвал f и ждет его конца
- f выполнен
- теперь g продолжил работать
- теперь h продолжил работать
- теперь main продолжил работать

## main

- запустился первым,
- закончил работать последним

## f

- запустился последним
- закончил работать первым

# Стек вызовов

## Множество вызовов – *стек*:

- Это коллекция объектов, в которую можно добавлять и из которой удалять элементы можно только с одного конца.
- Метафоры:
  - » Обойма
  - » Парковка в узком длинном тупике

*Стеки сами используются для построения алгоритмов, но это – следующий семестр.*

# Ошибемся – поздороваемся со стеком вызовов. Пример

```
public static void f() {  
    System.out.println("f");  
    int x = 10 / 0;  
}  
public static void g() {  
    f();  
    System.out.println("g");  
}  
public static void h() {  
    g();  
    System.out.println("h");  
}  
public static void main(String[] args) {  
    h();  
    System.out.println("main");  
}
```

# Красотища

```
Exception in thread "main" f
java.lang.ArithmeticException: / by zero
    at MyClass2.f(MyClass2.java:32)
    at MyClass2.g(MyClass2.java:36)
    at MyClass2.h(MyClass2.java:41)
    at MyClass2.main(MyClass2.java:46)
    at
    sun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)
        at
    sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethod
AccessorImpl.java:62)
        at
    sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegati
ngMethodAccessorImpl.java:43)
        at
    java.lang.reflect.Method.invoke(Method.java:497)
        at
    com.intellij.rt.execution.application.AppMain.main(AppMa
in.java:144)
```

# Красотища

Exception in thread "main" f  
 java.lang.ArithmeticException: / by zero  
     at MyClass2.f(MyClass2.java:32)  
     at MyClass2.g(MyClass2.java:36)  
     at MyClass2.h(MyClass2.java:41)  
     at MyClass2.main(MyClass2.java:46)  
     at  
 sun.reflect.NativeMethodAccessorImpl.invoke0(Native  
 Method)  
     at  
 sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethod  
 AccessorImpl.java:62)  
     at  
 sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegati  
 ngMethodAccessorImpl.java:43)  
     at  
 java.lang.reflect.Method.invoke(Method.java:497)  
     at  
 com.intellij.rt.execution.application.AppMain.main(AppMa  
 in.java:144)

# Параметры функции

- Нет «var», «&», «\*» и др.
- Все примитивные типы передаются как *параметры-значения* (их значение копируется в вызов метода).
- Все ссылочные типы данных – *по ссылке, параметр-переменная*
  - При этом очевидно, сама ссылка («адрес») ведет себя как параметр значения

# Пример

```
public static void inc(int x) {  
    x += 1;  
}
```

```
public static void main(String[] args) {  
    int x = 2;  
    inc(x);  
    // x = ?  
}
```



# Процедура

## Обратите внимание

Для ссылочных типов `void` методы могут быть функциональны  
для изменения содержимого

```
public static void fillArrayByRandomIntegers(int[] array) {  
    Random random = new Random();  
    for (int i = 0; i < array.length; i++) {  
        array[i] = random.nextInt();  
    }  
}
```

# Ho

```
public static void newArray(int [] a) {  
    int [] b = new int[]{1,2,3};  
    a = b;  
}  
  
public static void main(String[] args) {  
    int [] z = new int[]{1,1,1};  
    newArray(z);  
    // z[2] = ?  
}
```

# Полезные штуки для массивов - `java.util.Arrays`

Вывод массива на экран одной строчкой:

```
System.out.println(Arrays.toString(массив));
```

Сравнение массивов

```
Arrays.equals(массив1, массив2)
```

Сортировка (быстрая, Dual-Pivot)

```
Arrays.sort(массив)
```

Копирование массива

```
Arrays.copyOf(массив1, массив2)
```

...

*Еще много методов, позволяющих делать то, что обычно приходится делать вручную циклом.*

# Примитивы по ссылке. В Java такого нет, поэтому C#

- По значению

```
static int Max(int a, int b)
{
    return a > b ? a : b;
}
...
int y = Max(n, 10);
```

- По ссылке

```
static void Swap(ref int a, ref int b)
{
    int t = a;  a = b;  b = t;
}
...
Swap(ref x, ref y);
```

# Сигнатура метода

**название, список типов параметров**

**•—возвращаемый тип (не входит!)**

По сигнатуре в момент вызова определяется, какой метод использовать (есть ли он вообще)

«связывание»

# Перегрузка метода

Методы должны различаться по сигнатуре! По названию – не обязательно!

**Перегрузка** - объявление методов с одинаковыми именами, но разными наборами параметров

- Работает потому, что сигнатуры разные

# Пример

```
public static double difference(double a, double b) {  
    return Math.abs(a - b);  
}  
  
public static double difference(double a, double b, double e)  
{  
    double result = Math.abs(a - b);  
    return result > e ? result : 0;  
}  
  
public static void main(String[] args) {  
    System.out.println(difference(2.01, 2.0));  
    System.out.println(difference(2.01, 2.0, 0.00001));  
}
```

# Такая перегрузка плохо

```
public static void f1(double a) {  
    System.out.println("double");  
}
```

```
public static void f1(int a) {  
    System.out.println("int");  
}
```

Работает, да. Но... ?



# А такая перегрузка не работает

```
public static void f1(int a) {  
    System.out.println("void");  
}  
public static int f1(int a) {  
    return a + 1;  
}
```

Почему?

# Рекурсия

- «вызов функцией самой себя»
- Имеет серьезное математическое основание
  - рекурсивные функции – альтернатива Машине Тьюринга для задания функций
  - 3 базовых функции, 3 операции
- Со школы помним «рекуррентные соотношения».

# Что надо уметь делать

- Описывать, в чем рекурсивность задачи

+

- Уметь явно указывать границу (когда рекурсия останавливается)

# Пример #1. Натуральные числа

- Рекурсивное определение
  - 1 – натуральное число
  - Если  $x$  – натуральное, то  $x + 1$  – натуральное

# Пример #2. Факториал

- $\text{Факториал}(0) = 1$
- $\text{Факториал}(n) = n * \text{факториал}(n-1)$

# Пример #3. Группа людей

- Рекурсивные определения имеют место и в реальном мире.
- Везде, где есть итерация, можно применить рекурсию.
  - Два человека – группа людей
  - Группа людей + человек – снова группа людей

# Факториал (рекурсивно)

```
public static int fact(int n) {  
  
    if (n == 0) {  
        return 1;  
    } else {  
        return n * fact(n - 1);  
    }  
  
}
```

# Поиск максимума в массиве

```
public static int maxOfArray(int[] array, int k) {  
    if (k == array.length - 1) {  
        return array[k];  
    } else {  
        int m = maxOfArray(array, k + 1);  
        return m > array[k] ? m : array[k];  
    }  
}
```

...

```
System.out.println(maxOfArray(array, 0));
```



# Если не указать границу рекурсии

– что произойдет?

```
public static int fact(int n) {  
    return n * fact(n - 1);  
}
```

# Stack Overflow

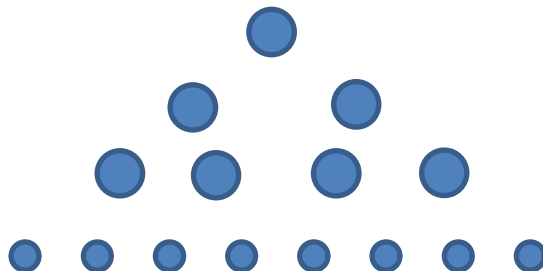
- Переполнение стека вызовов
  - «Довывывался»
- Максимальное значение можно настраивать в JVM (НО НЕ НУЖНО!)
  - Эксперименты давали 7000-8000.

# Рекурсия и циклы

- Все циклы **можно смоделировать** с помощью рекурсии!
  - Все-все. Но есть нюансы:
    - Цикл “while(true)” – это ... ?
    - Еще нюансы (!)
- При этом она не должна проигрывать по сложности
  - См. далее

# Bad Fibonacci

```
public static int fib(int n) {
    if (n == 1 || n == 2) {
        return 1;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}
```



! А как лучше. Подсказка: «Мемоизация», замена  $fib(n - 1)$  на  $fib[n - 1]$

# Рекурсия -> цикл

**Переписать** рекурсивный алгоритм в нерекурсивный можно, только если рекурсия **хвостовая** (**рекурсивный вызов – «последняя» операция!**)

Написанный нами факториал – не хвостовая (но можно сделать хвостовой).

```
public static int fact(int n) {  
  
    if (n == 0) {  
        return 1;  
    } else {  
        return n * fact(n - 1);  
    }  
  
}
```

Это не хвостовая рекурсия (после рек.вызова еще идут операции).

# Пример хвостовой рекурсии

```
public static void print0n (int n) {  
    System.out.println(0);  
    if (n > 1) {  
        print0n(n - 1);  
    }  
}
```

```
while (n > 1) {  
    System.out.println(0);  
    n -= 1;  
}  
System.out.println(0);
```



**ПОГОВОРИМ О 2Х ТЕМАХ, КОСВЕННО СВЯЗАННЫХ  
С ФУНКЦИЯМИ**

$$x^2$$

```
double x2 = x * x;
```

**ИЛИ**

```
double x2 = Math.pow(x, 2);
```





В чем прелесть работы с ними на компьютере?

# ВЕЩЕСТВЕННЫЕ ЧИСЛА

# Вещественные числа

Их необходимо считать только до определенной точности (дальше не нужно)

*калькулятор на экране имеет место только для 10 символов – так зачем считать дальше 9го знака после запятой?*



# Предел последовательности

$$\forall \varepsilon > 0 \exists n = n(\varepsilon) \in \mathbb{N} \forall n > n(\varepsilon) (|a_n - A| < \varepsilon)$$

Тогда  $A$  – предел последовательности  $a_n$

*Но что это реально означает?*

$$\forall \varepsilon > 0 \exists n = n(\varepsilon) \in \mathbb{N} \forall n > n(\varepsilon) (|a_n - A| < \varepsilon)$$

Раскроем модуль, получим:

$$A - \varepsilon < a_n < A + \varepsilon$$

Простым языком – числа  $a_n$  находятся **недалеко** от  $A$   
(**примерно ему равны с точностью  $\varepsilon$** )

- Примерно !?

- Для любого  $\varepsilon$  !?

$$\forall \varepsilon > 0 \exists n = n(\varepsilon) \in \mathbb{N} \forall n > n(\varepsilon) (|a_n - A| < \varepsilon)$$

# Возьмем простую последовательность




- $a_n = \frac{1}{n}$
- $\lim_{n \rightarrow \infty} a_n = 0$

# Вычислим

- $a_1 = 1$
- $a_2 = 0,5$
- $a_3 = 0,33333333333333$
- $a_4 = 0,25$
- $a_5 = 0,2$
- $a_6 = 0,16666666666667$
- $a_7 = 0,1428571428571429$
- $a_8 = 0,125$
- ...
- ...
- $a_{100} = 0,01$
- ...
- $a_{10000} = 0,0001$
- ...
- $a_{1000000} = 0,000001$
- ...
- $a_{1000000000000000} =$   
 $0,000000000000001$



# А теперь вспомним $(-\varepsilon < a_n - A < \varepsilon)$

- ...
- $a_{100} = 0,01 - 0 = \mathbf{0,01}$   **это  $\varepsilon$**
- ...
- $a_{10000} = 0,0001 - 0 = \mathbf{0,0001}$  
- ...
- $a_{1000000000000000} = 0,0000000000000001 - 0 = \mathbf{0,0000000000000001}$  
- **это  $n(\varepsilon)$**   $a_n - A$

# МАТАН VS ИНФОРМАТИКА

- Математический анализ говорит **о любом  $\varepsilon$**  – какое бы малое мы не взяли, все равно будет последовательность «стремиться» (быть ближе) к числу  $A$ , быть похожей на него.
  - Ну и там на бесконечности будет «равна  $A$ »
- В программировании – а зачем нам эта бесконечность? Мы наоборот фиксируем  $\varepsilon$  – и это наша точность вычислений. Мы вычисляем приближенное значение - до тех пор, пока ...
  - Стоп, но мы же можем не знать предела...

# Ряды

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

$$\operatorname{sh} x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + \frac{x^{2n-1}}{(2n-1)!} + \dots$$

$$\operatorname{ch} x = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + \frac{x^{2n}}{(2n)!} + \dots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + (-1)^{n-1} \frac{x^{2n-2}}{(2n-2)!} + \dots$$

$$(1+x)^m = 1 + \frac{m}{1!}x + \frac{m(m-1)}{2!}x^2 + \frac{m(m-1)(m-2)}{3!}x^3 + \dots$$

$$\frac{1}{1+x} = 1 - x + x^2 - x^3 + \dots + (-1)^{n-1}x^n + \dots$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots + (-1)^{n-1} \frac{x^n}{n} + \dots$$

$$\operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)} + \dots$$

$$\arcsin x = x + \frac{1}{2} \frac{x^3}{3} + \frac{1 \cdot 3}{2^2 2!} \frac{x^5}{5} + \frac{1 \cdot 3 \cdot 5}{2^3 3!} \frac{x^7}{7} + \dots$$

$$\operatorname{tg} x = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \dots + \frac{2n-2}{(2n-1)!}x^{2n-1} + \dots$$

# Пример

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

С каждым  $n$  слагаемое все меньше и меньше  
(не влияет на сумму)

# Вычисление $e^1$

$$a_0 = 1$$

$$s_0 = a_0 = \mathbf{1}$$

$$a_1 = 1$$

$$s_1 = s_0 + a_1 = 1 + 1 = \mathbf{2}$$

$$a_2 = 1/2! = 0,5$$

$$s_2 = s_1 + a_2 = 2 + 0,5 = \mathbf{2,5}$$

$$a_3 = 1/3! = 1/6 = 0,16666667$$

$$s_3 = \mathbf{2,6666666667}$$

$$a_4 = 1/4! = 1/24 = 0,0416666666666667$$

$$s_4 = s_3 + a_4 = \mathbf{2,70833333333333}$$

$$a_5 = 1/5! = 1/120 = 0,0083333333333333$$

$$s_5 = \mathbf{2,71666666666666}$$

$$a_6 = 1/6! = 1/720 = 0,0013888888888888$$

$$s_6 = \mathbf{2,71805555555555}$$

.....

2.71825396825

2.71827876984

2.71828152557

2.71828180115

2.7182818262

2.71828182829

2.71828182845

2.71828182846

# Math.pow(x, y)

- $x^y = e^{\ln(x^y)} = e^{y \ln(x)}$
- *Math.exp(y \* Math.ln(x));*
- Когда вы вызываете Math.exp или Math.log, там происходят такие же вычисления.
- *Поэтому Math.pow(x, 2) – very, very bad*  
– *примерно 100 раз медленней x \* x*

