# Гуляев Г.М.

# Математика и программирование

## Язык Julia для математика

Экспериментальная математика

# Вычисление как исследование

- Со времен Эйлера (1707-1783) вычисления и эмпирический анализ являются в математике важным методом исследования
- Современные компьютеры при вычислениях дают огромное преимущество
- $F_n = 2^{2^n} + 1$ - простые? Вопрос Ферма.

  3, 5, 17, 257, 65537, 4294967297, 18446744073709551617, ...
- Эйлер нашел: $F_5 = 641 \cdot 6700417$
- $k \cdot 2^{n+1} + 1$ - вид делителей (Эйлер), то есть проверялись только числа вида $k \cdot 64 + 1$ и при $k=10$ все получилось.
- На языке **Julia** достаточно выполнить команды:

```
list = map(n -> big(2)^2^n+1, 0:6)
map(x -> factor(x),list)
```

# Выбор языка для математика

- **Проприетарные продукты Mathematica (язык Wolfram) и Matlab (язык MATLAB)**
- **Функциональный язык программирования Haskell**
- **Интерпретируемый язык R для стат. обработки**
- **Относительно новый язык программирования Julia**
- **Универсальные языки (C,C++, Java, Scala, Python,...)**
- **Я выбрал для себя Julia по следующим причинам:**

1. свободный язык, ориентированный на математику
2. универсальный, высокоуровневый язык
3. производительность на уровне C, C++
4. процедурная и функциональная парадигма
5. готовые библиотеки для математических функций

# Краткий обзор языка Julia

- **Рассмотрим примеры и отметим отличия от Scala**
- **Тип переменной определяется во время выполнения:**

```
f(x) = sum(x)


f(3)  # 3
f(5.6) # 5.6
f([1,2,3,4])  # 10
f((1,2,3)) # 6
f((1,2,"a")) # Ошибка
f("abc") # Ошибка
x = 1291821982 # Int64
x = 129182198219829182198291283434 # Int128
x = 1291821982198291821982912834341212123453223 # BigInt
```

# Функции в Julia

- **Многострочная функция (сумма чисел < n, делящихся на 3 или 5):**

```
function f(n)
    s = 0
    for x in 1:n-1
        if x%3==0||x%5==0
            s+=x
        end
    end
    s
end
```

- **Та же функция в функциональном стиле:**

```
f(n) = sum(filter(x -> x%3==0||x%5==0, 1:n-1))
```

- **Рекурсия (факториал):**

```
fact(n) = if n<2 1 else n*fact(n-1) end
```

# Модули и построение графиков

- **Подключение модулей**

```
using Primes
```

**если модуль отсутствует, то предлагается его скачать**

```
import Pkg; Pkg.add("Primes")
```

- **Примеры на построение графиков:**

```
using Plots
plot(1:10)
plot(sin, -pi, pi)
plot([sin,cos], -pi, pi)
f(x) = x^2
plot([sin,cos,f], -pi, pi)
r(θ) = 1 + cos(θ)
plot(r, 0, 2π, proj=:polar)
```

- **Примеры 3D графиков: plot3d**

# Векторы и матрицы

- **Вектор - одномерный массив**

```
a  = [1,2,3,4] #  a[1] = 1, a[2] = 2, … , a[1][1] = 1, a[2][1] = 2
a = Vector{Int64}() # пустой вектор
a = [1 2 3 4] #  a[1] = 1, a[2] = 2, … , a[1,1] = 1, a[1,2] = 2
a = [1 2 3; 4 5 6] # матрица 2x3, a[2,3] = 6
reshape(a,6) # вектор [1,4,2,5,3,6]
a' # транспонированная матрица 3x2: [1 4; 2 5; 3 6]
a*a' # произведение матриц: [14 32; 32 7]
[1 2 3 4]*[5 6 7 8]' # скалярное произведение векторов (70)
for x in [1,2,3,4] println(x) end
```

- **Изменение массива:**

```
push!([1,2,3,4],5) # [1,2,3,4,5]
pushfirst!([1,2,3,4],5) # [5,1,2,3,4]
insert!([1,2,3,4],2,5) # [1,5,2,3,4]
sort!([5,2,4,1,3]) # [1,2,3,4,5]
[1,2,3,4].^2 # [1,4,9,16] равносильно map(x -> x^2, [1,2,3,4])
```

# Производительность (сравнение со Scala)

- **Задача:**

Число 145 интересно тем, что оно равно сумме факториалов своих цифр ( **145 = 1! + 4! + 5!** ). Найти все такие числа.

- **Решение:**

```
map(x -> factorial(x),0:9) # [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
```

то есть сумма факториалов цифр n-значного числа <= n*362880

3000000>362880*7 имеет смысл рассматривать числа не более 3000000

```
sf(n) = sum(map(x -> factorial(x),digits(n)))
filter(x -> x==sf(x),1:3000000) # [1,2,145,49585]
```

- **Найдем сумму всех n из 1:30000000 таких, что sf(n)|n**

```
sum(filter(x -> x%sf(x)==0,1:30000000)) # 179550168607
```

- **Затраченное время:**

```
@time sum(filter(x -> x%sf(x)==0,1:30000000)) # 7.303116 seconds
```

- **То же самое на Scala: 23.332 с**

# Делители числа, дружественные числа

- **Нахождение собственных делителей n:**

```
function divs(n)
  0 < n || throw(ArgumentError("Ожидалось n > 0, а не $n"))
  1 < n || return Vector{Int64}()
  !isprime(n) || return [1]
  d = [1]
  for (p, k) in factor(n)
    c = [p^i for i in 0:k]
    d = d*c'
    d = reshape(d, length(d))
  end
  sort!(d)
  return d[1:end-1]
end
```

# Выводы

- **Плюсы языка Julia:**
  1. Простой удобный синтаксис
  2. Производительность (в 2,5-3 раза быстрее **Java** или **Scala**)
  3. Поддержка математики (функции, матрицы, графики и тп)
  4. Удобный REPL (read-eval-print loop)
- **Минусы:**
  1. Менее удобно, чем в **Scala**, писать однострочные функции из-за блока **begin … end** вместо **{ … }**
- **Язык Julia - отличное средство для математика, который готов научиться программированию на нем**

# Спасибо за внимание !

www.altailand.ru